SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

VISUAL BASIC 6.0 OHJELMIEN MIGRAATIO JA PÄIVITYS VB.NET: IIN

Ohjelmien migraatio, lisensointisuojaus ja asennusohjelmien luonti

TEKIJÄ/T: Valtteri Kovalainen
Eetu Häkkinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Valtteri Kovalainen, Eetu Häkkinen	
Työn nimi Visual Basic 6.0 ohjelmien migraatio ja päivitys VB.NET:iin	
Päiväys 26.7.2019	Sivumäärä/Liitteet 56
Ohjaaja(t) lehtori Veijo Pitkänen, lehtori Mikko Pääkkönen	
Toimeksiantaja/Yhteistyökumppani(t) Insinööritoimisto Laaturakenne Oy	

Tiivistelmä

Opinnäytetyön tarkoituksena oli päivittää yhdeksän Visual Basic 6.0:lla tehtyä sovellusta .NET-ympäristölle, korjata tai uudelleenrakentaa päivityksessä rikkoutuneet toiminnot, luoda sovelluksille asennusohjelmat ja lisätä sovelluksiin lisensointisuojaus. Sovelluksia käytetään rakennusinsinööritöissä vastaan tulevien mittausten laskemisessa, ja siitä saatujen tulosten visualisointiin, taulukointiin, tallentamiseen ja tulostamiseen.

Opinnäytetyö jaetaan kolmeen osaan: Migraatio, asennusohjelmien luonti, ja lisensointi. Migraatio-osuudessa sovellukset päivitettiin pois Visual Basic 6.0 -ympäristöstä .NET-ympäristöön, jotta niiden jatkokehitys pysyisi mahdollisena ja nykyaikaisempana. Tämä prosessi rikkoi monia alkuperäisten ohjelmien toimintoja, pääosin taulukot, kaavionpiirrot, ja tulostamisen, joten seuraava vaihe oli niiden korjaaminen tai uudelleenrakentaminen. Kun ohjelmat saatiin migraation jälkeen takaisin toimintakuntoon, aloitettiin lisensoinnin ja asennuspakettien rakentaminen. Lisensoinnissa käytettiin Soracon tarjoamaa Quick License Manager -palvelua, jolla sovelluksiin lisättiin lisenssisuojaus. Tämä estää sovellusten luvattoman käytön. Asennuspaketit rakennettiin Inno Setup-sovelluksen avulla ja niiden avulla käyttäjä voi asentaa sovellukset tietokoneellensa kätevämmin.

Opinnäytetyöstä saatiin suoritettua kaikki asetetut tavoitteet, ja toimeksiantaja ottaa päivitetyt sovellukset käyttöön yhtä lukuun ottamatta, sillä sitä jatkokehitetään vielä. Sovelluksille oli myös muita toimintoja, joita olisi voitu lisätä, kuten esimerkiksi kielenvalinta, laskuyksiköiden kertoimien vaihto ja yksikkötestit laskufunktioille, mutta niiden ei todettu olevan tarpeeksi kriittisiä tärkeydeltään verrattuna opinnäytetyössä määriteltyihin tehtäviin.

Avainsanat
Visual Basic, .NET, Migraatio, Päivitys, Lisensointi

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Valtteri Kovalainen, Eetu Häkkinen			
Title of Thesis Updating and Migrating Visual Basic Applications to VB.Net			
Date	23 September 2019	Pages/Appendices	56
Supervisor(s) Mr Veijo Pitkänen, Senior Lecturer, Mr Mikko Pääkkönen, Senior Lecturer			
Client Organisation /Partners Laaturakenne Oy			

Abstract

The main purpose of this thesis was to update nine Visual Basic 6.0 applications to use the .NET environment. This included fixing or rebuilding any functionality that was broken in the update process, adding license protection to the applications and finally creating installation packages to make installation easier for the end user. The applications are used to calculate measurements of parts and materials used in construction. In addition, they can also visualise, print and save the results.

The thesis can be divided into three parts: Migration, creating the installation packages and licensing. Through migration the applications were moved from Visual Basic 6.0 to use Visual Basic .Net in order to better enable further development and keep them up to date. During this process many of the original functionalities of the programs were broken, the most impactful ones being tables, diagrams and printing. Due to this, these functionalities had the top priority and they were fixed or rebuilt completely on a case by case basis. After the programs had all been migrated and fixed, focus was shifted to licensing and installation. The licensing was done to prevent the unauthorised use of the programs, and it was implemented using Quick License Manager, software made by Soraco Technologies. The installation Packages were made using Inno Setup by JRSoftware.

As a result, all the goals set for the thesis were reached and the commissioner will start using eight of the nine migrated applications. The remaining application will resume development that was put on hold by the commissioner to allow migration. There were also other possible functionalities that could have been added to the programs, such as language selection, saved custom factors for the calculations and testing for calculational errors between versions. These were considered to be too time consuming and not important enough compared to the other implemented features, so they were left out.

Keywords
Visual Basic, .NET, Migration, Updating, Licensing

1 JOHDANTO 6
1.1 Lyhenteet ja määritelmät 6
1.2 Työn tilaaja, työkalut ja tekijänoikeudet 7
2 MIGRAATIOPROSESSI 8
2.1 Taustatietoa migraatiosta 8
2.2 Siirtyminen .NET -ympäristöön 9
2.3 Koodin muutokset siirtymisen myötä 9
2.3.1 Laskennan oikeellisuus 11
2.4 Microsoft Power Packs 11
2.5 Täysin uudet toiminnot 12
2.5.1 DotFormat-funktio 12
2.5.2 StringFormat -funktio 13
2.5.3 Taulukot 13
2.5.4 Resoluution skaalaus 17
2.5.4.1 Skaalaus dynaamisissa elementeissä 18
2.5.5 Ohjelman sulkeminen 19
2.6 Uudelleenkirjoitetut funktiot ja toiminnot 19
2.6.1 Tulostus 19
2.6.1.1 MigraDoc ja Uuden tulostusfunktion rakenne 20
2.6.1.2 Dokumentin luonti 21
2.6.1.3 Sisällön kirjoitus 22
2.6.1.4 Vierekkäiset taulukot 25
2.6.1.5 Sivunvaihto 25
2.6.1.6 Dokumentin tulostaminen 26
2.6.2 Piirto 27
2.6.2.1 Piirron sovellukset 28
2.6.2.2 Käyrien piirto tarkemmin 30
2.7 Pois jätetyt toiminnot 32
2.7.1 Alkuperäinen taulukointi 32
2.7.2 Käyttöliittymien piilottaminen ja sulkeminen 32
2.8 Migraation lopputulos 33
3 ASENNUSOHJELMA 34

3.1 Inno Setup 34
3.2 Asennusohjelman luonti 35
3.3 Asennusohjelman lopputulos 37
4 LISENSOINTI 38
4.1 Lisenssiohjelma 38
4.2 Quick License Manager 38
4.2.1 Käyttöönotto 40
4.2.2 Hallintasovellus ja aloittaminen 40
4.2.3 Tuotteen määrittely 41
4.2.4 Sovelluksen suojaus 43
4.2.5 Avainten luonti 47
4.2.6 Avainten vahvistus 49
4.2.7 Muutokset ja lisäykset ohjelmistoihin 50
4.2.7.1 Lisenssin tarkistus 50
4.2.7.2 Rekisteröinti 53
4.2.7.3 LicenseValidator-luokka 54
4.2.8 QLM:in käyttö 55
4.3 Lisensoinnin lopputulos 55
5 JATKOKEHITYS JA POHDINTA 57
5.1 Pohdinta – Eetu Häkkinen 57
5.2 Pohdinta – Valtteri Kovalainen 57
5.3 Jatkokehitys 58
5.3.1 Käännösmahdollisuus 58
5.3.2 Muokattavat vakioarvot ja -kertoimet 59
5.3.3 Laskennan automaattinen testaus 59
6 LÄHDELUETTELO 60

1 JOHDANTO

Työn tarkoituksena oli siirtää yhdeksän samankaltaista Visual Basic 6.0:lla tehtyä sovellusta uudempaan Visual Basic .Net -ympäristöön. Työn tilasi insinööritoimisto Laaturakenne Oy, jossa kontaktihenkilönä toimi toimitusjohtaja Ari Korhonen. Hän myös tarjosi teknistä tukea ja tietoa, sillä ohjelmistot ovat alun perin Korhosen itsensä kirjoittamia.

Työn tekijöinä olivat Eetu Häkkinen ja Valtteri Kovalainen, ja opinnäytetyö suoritettiin yhteistyönä. Edistymisestä pidettiin palavereita ja kirjoitettiin viikkoraportteja säännöllisin väliajoin, mutta muuten työskentely tehtiin itsenäisesti. Työaikatauluksi sovittiin, että työtä tehdään arkipäivisin, kun taas viikonloput pidetään vapaata.

Opinnäytetyö koostui kolmesta pääaiheesta, jotka olivat sovellusten migraatio, lisensointi ja asennusohjelman luonti. Ensimmäisenä tehtiin yhdessä migraatio. Tämä vei suurimman osan ajasta, sillä se oli työvaiheista työläin. Kaksi myöhempää vaihetta jaettiin niin, että Kovalainen hoiti lisensoinnin, kun taas Häkkinen huolehti asennusohjelmasta. Aiheet käydään raportissa läpi siinä järjestyksessä, jossa työ alun perin tehtiin, eli ensin käydään läpi migraatio ja mitä siihen kuului, josta siirrytään asennusohjelmaan, jonka jälkeen käydään läpi lisensointia.

1.1 Lyhenteet ja määritelmät

- VB – Visual Basic on syntynsä vuonna 1991 saanut Microsoftin kehittämä ohjelmointikieli.
- VB6.0 – Visual Basicin versio, joka on julkaistu 1998.
- VB.NET – Vuonna 2002 julkaistu versio, jossa siirryttiin Microsoftin .Net ympäristöön. Poikkeaa rakenteeltaan ja toiminnallisuudeltaan aiemmista versioista, muistuttaen enemmän muita .NET-kieliä.
- Visual Studio - Microsoftin ohjelmankehitysympäristö, jota voidaan käyttää useiden kielien kanssa. Tässä työssä sitä on käytetty VB.NET-kehitykseen.
- .NET Framework – Myöskin Microsoftin kehittämä ohjelmakomponenttikirjasto, käytetään Visual Studiolla kehitetyissä ohjelmistoissa. Tukee noin 20 eri ohjelmointikieltä, hoitaa suurimman osan ohjelmistojen perustoiminnoista automaattisesti.
- NuGet – Em. kehitysympäristön paketinhallintaohjelma, jolla voi ladata ohjelmakirjastoja ja -paketteja.
- Ohjelmakirjasto – Luokka, kokoelma toimintoja, tai muu lisäosa, jota ohjelma käyttää hyväkseen.
- Funktio/Proseduuri – Tunnetaan myös nimellä aliohjelma. Se on osa ohjelmaa, joka ajetaan vain, kun sitä kutsutaan pääohjelmasta tai toisesta aliohjelmasta. Yksinkertainen aliohjelma voisi olla esimerkiksi sellainen, johon annettaisiin kaksi lukua, jotka sitten laskettaisiin yhteen ja palautettaisiin yhteenlaskun tulos.
- Objekti/Olio – Olio-ohjelmoinnin perusta, sisältää toimintoja ja muuttujia.
- Julistaminen – Uuden olion luonti luokasta. Julistaessa määritetään mahdollisesti olioon tiettyjä arvoja. Luokassa voi olla erikseen määritelty, mitä julistettaessa tapahtuu, jos mitään.

- QLM – Quick License Manager on Soracon kehittämä lisensointiratkaisu, joka mahdollistaa piratismin ainakin osittaisen estämisen suhteellisen pieneen hintaan.
- Inno Setup – Ilmainen Jordan Russellin ja Martijn Laanin kehittämä asennusohjelman luoja, joka mahdollistaa helpon luonnin kehittäjälle ja vaivattoman asennuksen käyttäjälle.
- MigraDoc – Ilmainen ohjelmakirjasto, jota tässä työssä on käytetty koodissa luodun tekstin tulostamiseen tai pdf-muotoon tallentamiseen.
- Winforms – Lyhenne sanoista Windows Forms, graafinen kirjasto .NET -ympäristössä. Käytetty käyttöliittymien luomiseen.
- Kuormittaminen – Englanniksi overloading. Tapa luoda useita eri funktioita samalla nimellä, mutta eri toiminnallisuuksilla.

1.2 Työn tilaaja, työkalut ja tekijänoikeudet

Työn tilasi Savonian kautta insinööritoimisto Laaturakenteen toimitusjohtaja Ari Korhonen. Kaikki työstä tuotettu koodi kuuluu Laaturakenteelle, sillä näin sovittiin heti työn alkaessa. Kaikki käytetyt ohjelmakirjastot ja ohjelmistot ovat joko täysin ilmaisia, tai niihin on ostettu asianmukaiset lisenssit.

Ohjelmistojen migraatiossa ja kehittämisessä käytettiin yksinomaan Microsoftin Visual Studio -kehitysympäristöä. Asennusohjelma luotiin ilmaisella Inno Setupilla ja lisensointi tehtiin Soracon Quick License Managerilla, josta ostettiin työn myötä Laaturakenteelle Express-versio.

2 MIGRAATIOPROSESSI

2.1 Taustatietoa migraatiosta

Opinnäytetyön ensimmäinen ja työläin vaihe oli sovelluksien siirtäminen Visual Basic 6.0 -kielestä modernimpaan .NET -pohjaiseen Visual Basiciin, lyhennettynä VB.NET.

Migraatiolla tarkoitetaan sovellusten siirtämistä ympäristöstä toiseen. Syitä migraation suorittamiselle ovat esimerkiksi siirtyminen uusille alustoille, tehokkuuden parantaminen tai uudempaan päivittäminen. Tässä tapauksessa kyse oli jälkimmäisestä, ja migraatio tehtiin, jotta yhteensopivuus ja jatkokehitys onnistuvat paremmin uudempien järjestelmien kanssa. (Verticom Oy, 2018.)

Migraatiossa huomattiin useampi korjattava ongelma, joista suurempia oli kaksi. Vanhat ohjelmistot toimivat niin, että tietokoneen desimaalierottimeksi oli manuaalisesti asetettava piste, ellei niin ollut asetettu jo valmiiksi. Jos tämän jätti tekemättä, ohjelmat eivät osanneet lukea, tulostaa tai laskea dataa oikein. Uudempiin versioihin siis päivitettiin toiminnallisuus siten, että dataa pystyy syöttämään, oli desimaalierotin piste tai pilkku koneen asetuksista riippumatta. Ohjelmistot muuntavat tämän aina pistettä käyttävään muotoon, joka näytetään tulosteissa ja ohjelmassa tuloksia tarkasteltaessa. Toiseksi ongelmaksi osoittautuivat ns. legacyfunktiot, eli vanhat funktiot, joita Visual Studio ei osannut automaattisesti muuntaa uudempaan muotoon, joten niille piti luoda tai etsiä korvaavat toiminnot.

Ohjelmistoja on yhdeksän erilaista, jotka voidaan jakaa useammalla eri tavalla. Ensimmäinen ryhmittelytavoista ohjelmille on laskettavan kappaleen tyyppi, eli joko palkki tai poikkileikkaus. Toinen tapa on jakaa ne materiaalin mukaan jänne- tai teräsbetonia käyttäviksi. Kolmas tapa, jota käytettiin työnjaon tekemiseen, on jakaa ohjelmistot laskentastandardin mukaan. Kovalainen hoiti vanhempien suomalaisten standardien mukaan tehtyjen ohjelmien migraation, kun taas Häkkiselle jäi uudempien EU-standardien mukaisten ohjelmistojen migraatio. Ominaisuuksiltaan laskentastandardeilla erotetut ohjelmat ovat samanlaisia, mutta ne käyttävät erilaista laskulogiikkaa ja saattavat sisältää pieniä eroja toiminnallisuuksissa. Viimeiseksi jäi yksi EU-standardia seuraava ohjelmisto, joka on kaikista suurin, uusin ja monipuolisin kyseessä olevista ohjelmistoista. Tämän ohjelman migraatio hoidettiin tiiviimmässä yhteistyössä, toisin kuin aiempien ohjelmien kanssa, vaikka niissäkin jaettiin funktioita ja dataa työn nopeuttamiseksi.

Migraatiossa käytetyistä työkaluista ainoastaan lisenssiohjelma Quick License Manager vaati lisenssin ostamisen. Asennusohjelmien luontiin käytetty Inno Setup oli täysin ilmainen, kuten myös kaikki käytetyt ohjelmakirjastot.

Migraatio aloitettiin tutkimalla eri keinoja VB6:lla tehtyjen lähdekoodien päivittämiseen Visual Basic 6.0 -ohjelmointiympäristöstä .NET-pohjaiseen ympäristöön. Parhaan keinon todettiin olevan lähdekoodin päivittämistä useamman ohjelmointiympäristön läpi, päivittäen koodia askel askeleelta (Nibu, 2011).

2.2 Siirtyminen .NET -ympäristöön

Migraatio aloitettiin avaamalla sovellusten projektitiedostot Visual Basic 2008 Express- ohjelmointiympäristössä. Ympäristö käänsi sovellukset automaattisesti VB6-muodosta uudempaan projektirakenteeseen. Tämä tehdään sen takia, koska Visual Basic 6.0:n projektirakenne ei ole yhteensopiva Visual Studio 2017:n kanssa, joka on projektin kohdeohjelmointiympäristö, ja Visual Basic 2008 Express on viimeisin Visual Studio -ohjelmisto, jossa on työkalu projektin siirtämiseen uudempaan muotoon.

Kun lähdekoodien projektirakenne oli päivitetty Visual Basic 2008 Express -projektimuotoon, päivitetyt lähdekoodit avattiin Visual Studio 2017:ssa. VS2017 päivitti projektirakenteet uudestaan ja samalla päivitti lähdekoodin Visual Basicista Visual Basic .Net -muotoon.

Kun lähdekoodit oli päivitetty halutulle ympäristölle, aloitettiin koodin käsin korjaus ja päivitys. Päivitysprosessi jätti osan koodista muokkaamatta, koska sille ei löydy vastiketta VB.NET:stä. Nämä koodin osuudet liittyivät suurimmilta osin tulostukseen, sekä käyrien ja taulukoiden piirtoon.

2.3 Koodin muutokset siirtymisen myötä

Kuten edellisessä osiossa on kerrottu, suurimmat muutokset koodissa tehtiin tulostukselle, käyrien piirrolle ja taulukoihin. Näistä löytyy raportista omat osionsa, joten tässä osuudessa keskitytään pienempiin muutoksiin.

Koodissa käytettiin Val-funktiota tekstin muuttamiseen numeroiksi. Tässä on mahdollisuus tulla tyyppieroja laskumoduuleissa, koska Val-funktio ottaa sopivimman tyypin numerolle, eikä se mahdollisesti sovi sille muuttujalle, johon Val-funktiolla arvoa syötetään. Siksi jokainen Val-funktio onkin ympäröity CSng-funktiolla, joka muuttaa syötetyn numeron Single-tyyppiseksi. Seuraavassa esimerkissä haetaan teksti kentästä ”Form7. TxtLVmuutos” ja muutetaan se Single-tyyppiseksi numeroksi muuttujaan Lvmuutos, kuten kuvasta 1 voidaan nähdä.

```
Lvmuutos = CSng(Val(Form7.TxtLVmuutos.Text))
```

KUVA 1. Muuttujan muuttaminen tekstistä Single-tyypin liukuluvuksi.

Toinen muokkaus on ollut kaikkien CStr-funktioiden korvaus Str-funktiolla. Molemmat tekevät saman toiminnon, eli muuttavat syötteen tekstiksi, mutta CStr-funktio käyttää tietokoneen asetuksia desimaalierottimelle. Koska kaikkien desimaalierottimien pitää olla piste riippumatta tietokoneen asetuksista, siirryttiin käyttämään Str-funktiota, joka käyttää aina pistettä desimaalierottimena. Jälkeenpäin huomattiin, että Str-funktiossa on omat ongelmansa, jotka ratkaistiin luomalla StringFormat-funktio.

Kaikille sovellusten ikkunoille tehtiin myös peruskorjauksia ja säätöjä. Kaikkien käyttöliittymien fontiksi asetettiin Arial ja fonttikooksi 10. Niihin lisättiin myös automaattinen vieritys. Käytännössä tämä toimii siten, että kun käyttäjä vaihtaa käyttöliittymän ikkunan kokoa, ilmestyy ikkunan sivuille automaattisesti vierityspalkit tarpeen mukaan.

Alkuperäisissä sovelluksissa tiedostontallennus- ja latausdialogit hoidettiin VB6:lle ominaisilla virheentarkistusmenetelmillä. Vanhassa koodissa asetettiin virheentarkistus funktioihin, joissa tarvitsee avata dialogi-ikkunaa joko tiedoston tallentamiseen tai avaamiseen. Jos dialogi-ikkuna suljetaan, se palauttaa virheen, jonka virheentarkistus huomaa, siten lopettaen funktion. Myös kaikki menetelmät, joilla kaikki dialogi-ikkunan avaaminen tehtiin, käyttivät VB6:lle ominaisia toimintoja, joten ne piti kaikki korvata vastaavilla toiminnoilla VB.NET:stä. Seuraavassa esimerkissä on vanha ja uusi dialoginavauskoodi tiedoston tallentamista varten. Muuttuja CommonDialog1Save on käyttöliittymään asetettu objekti, joka kysyy käyttäjältä tallennussijaintia dialogi-ikkunan muodossa.

```
'------------VANHA FUNKTIO-----------
'Alustetaan virheenkäsittely
On Error Resume Next

'Tarkkailaan Cancel-painiketta
CommonDialog1.CancelError = True

CommonDialog1.ShowSave

'Poistutaan avaamatta tiedostoa, jos painetaan Cancelia
If Err.Number = cdlCancel Then
    Exit Sub
End If


'------------UUSI FUNKTIO------------

'Tallennetaan lähtötiedot
CommonDialog1Save.Filter = "Teksti tiedostot (*.txt)|*.txt|Kaikki tiedostot (*.*)|*.*"
Dim DialResult As DialogResult = CommonDialog1Save.ShowDialog() 'Avataan dialogi-ikkunna, ja otetaan sen tiedot talteen DialResult-muuttujaan

'Poistutaan avaamatta tiedostoa, jos painetaan Cancelia
If DialResult = DialogResult.Cancel Then
    Exit Sub
End If
```

KUVA 2. Tiedoston tallentaminen, yllä vanha ja alla uusi metodi.

Uudessa funktiossa luodaan DialogResult-tyyppinen muuttuja DialResult, johon tallennetaan avatun tallennus- tai latausikkunan lopputulos. Kuten kuvasta 2 nähdään, jos tämä lopputulos on DialogResult.Cancel, eli käyttäjä on keskeyttänyt tallentamisen tai lataamisen, poistutaan funktiosta tekemättä mitään muuta.

Viimeiseksi migraatioprosessi jätti jälkeensä VB.NET:ssä hyödyttömäksi jääneitä osia. Näitä poistettiin sitä mukaan, kun niitä löytyi. Yleisimmät näistä olivat OvalShapeArray.vb-tiedosto ja LabelArray-komponentit joissain käyttöliittymissä. OvalShapeArray.vb-tiedoston alkuperäisestä tarkoituksesta ei ole tietoa, eikä sitä ole kutsuttu missään kohdassa koodia, joten oli loogista päätellä, että se on ylijäämää sovellusten kehitysasteelta. Tämä varmistettiin myöhemmin työn tilaajalta. Sovelluksista löytyi myös joistakin käyttöliittymistä LabelArray-tyyppisiä komponentteja. Näiden komponenttien tarkoitus oli toimia korvikkeina VB6:ssa vastaaville komponenteille. VB.NET:ssä ei näille ole muuta tarkoitusta tai toimivuutta, joten ne voitiin poistaa ilman ongelmia.

2.3.1 Laskennan oikeellisuus

Koska kyseessä on laskentaohjelmisto, oli äärettömän tärkeää, että laskennan logiikka ei muutu migraation aikana. Tämän suhteen ilmeni pieni määrä ongelmia, joista kenties suurimmat olivat rajaarvot ja niiden vertailu. Tietyissä laskentafunktioissa vertailtiin kahta eri arvoa, joista toinen oli kovakoodattu ja toinen otettiin muuttujasta. Tästä voidaan nähdä esimerkki kuvassa 3. Kuvassa on ehtolause, jossa tarkistetaan, että onko muuttuja Kuormataulukko3(i, 2) arvoltaan suurempi kuin 1.2.

```
If Kuormataulukko3(i, 2) > 1.2 Then
```

KUVA 3. Alkuperäinen vertailu

Visual Basic 6.0 ei ole tiukasti tyypitetty kieli, eli on normaalia julistaa muuttujia pelkkinä objekteina ja vertailla niitä vapaasti. Tämä ilmenee siitä, että alkuperäisessä ohjelmassa monet muuttujista on julistettu pelkästään objekteiksi, jonka jälkeen osaa niistä on käytetty tekstimuuttujien paikalla, kun taas osaa numeroiden paikalla, eikä tämä aiheuta ohjelmalle ongelmia. Numeroina käytettyjä muuttujia on voinut myös vertailla vapaasti keskenään haittaamatta toiminnallisuutta, vaikka niiden tyyppi ei ole ollutkaan sama. Ongelmana on, että .NET-ympäristö käsittelee eri tyypin muuttujia erilaisina. Kuvan 3 vertailulauseessa oikealla oleva luku 1.2 on oletuksena Double-tyyppinen, eli 64-bittinen liukuluku, kun taas vasemman puolen muuttuja on 32-bittinen liukuluku, eli tyypiltään Single. Kieli ymmärtää lukujen vertailun lähes kaikissa tilanteissa oikein, mutta lukujen ollessa yhtä suuret syntyy ongelma, joka aiheuttaa pieniä poikkeamia tuloksissa ja saattaa antaa väärän tuloksen (Microsoft Oy, 2015).

```
If Kuormataulukko3(i, 2) > CSng(1.2) Then
```

KUVA 4. Korjattu vertailu

Ongelma oli vakava ja erittäin vaikea jäljittää, mutta lopulta hyvin yksinkertainen korjata. Korjaus suoritettiin kuvan 4 näyttämällä tavalla muuttamalla kovakoodattu luku samaksi tyypiksi, kuin vertailtava muuttuja. Tällaisia vertailulauseita löytyi useampi ohjelmiston eri kohdista ja ne kaikki korjattiin samalla periaatteella.

2.4 Microsoft Power Packs

Ympäristönvaihto Visual Basic 6.0:sta Visual Studio 2017:ään hajotti monta toimintoa koodista ja sovelluksen ulkonäöstä. Moni graafinen ominaisuus hävisi migraation myötä, koska niille ei ollut vastiketta VB.NET-ympäristössä. Nämä graafiset ominaisuudet ovat erilaisia muotoja, viivoja ja ympyröitä, joita oli piirretty sovellusten eri käyttöliittymiin auttamaan tietojen syöttöä visuaalisesti. Jotta nämä muodot saatiin pidettyä käyttöliittymissä, sovelluksen viittauksiin tuli lisätä Microsoft.VisualBasic.Powerpacks. Power Packs sisältää Visual Basiciin useita toimintoja, joista erityisesti tarvitaan Line ja Shape -kontrollit, joiden avulla on mahdollista ylläpitää käyttöliittymiin piirrettyjä viivoja ja ympyröitä (Microsoft Oy, 2019).

Power Packs pitää ladata Microsoftin sivuilta sille tietokoneelle, jolla sovelluksia jatkokehitetään. Tarvittavat dll-tiedostot lisätään sovelluksen viittauksiin, joten käyttäjän ei tarvitse Power Packsia ladata tietokoneelle, jotta sovellukset toimisivat.

2.5 Täysin uudet toiminnot

2.5.1 DotFormat-funktio

Ohjelmaa varten täytyi kirjoittaa useita uusia toimintoja, joista osa korvaa tai parantaa aiempaa toiminnallisuutta ja osa lisää kokonaan uusia ominaisuuksia. Yksi näistä toiminnoista on DotFormat-funktio.

Yksi migraation vaatimuksista oli, että kaikissa ohjelmissa käytettäisiin desimaalierottimena pistettä riippumatta tietokoneen aika- tai sijaintiasetuksista. Tämä näkyisi vain käyttäjälle, joten koodin sisällä pystyttiin pilkkuja käyttämään tarpeen mukaan, kunhan ne muutetaan pisteiksi ennen näyttämistä käyttöliittymässä. Numeroita muotoillaan myös sopivan tarkoiksi ennen näyttämistä, joten se pitää tehdä samalla kun pilkkuja muutetaan pisteiksi. Tätä varten rakennettiin funktio, joka muotoilee syötetyn numeron haluttuun muotoon ja samalla muuttaa kaikki pilkut pisteiksi. Funktion nimeksi päätettiin antaa DotFormat, sillä sitä käytetään desimaalipisteen formatointiin. DotFormat ottaa vastaan kaksi parametria, jotka ovat nimeltään Input ja Form. Input on syöte, jota halutaan muokata, ja Form on muoto, johon Input halutaan muuttaa.

```
Public Function DotFormat(ByVal input As Single, ByVal form As String)
    Return Replace(Format(input, form), ",", ".")
End Function
```

KUVA 5. DotFormat-funktio, joka ottaa vastaan Single-tyyppisen muuttujan.

DotFormat yhdistää kaksi funktiota yhdeksi; Replace- ja Format-funktiot. Replace-funktion tarkoitus on etsiä syötettävästä muuttujasta, kuten tekstistä, halutut kohdat ja korvata ne halutulla muutoksella. Tässä tapauksessa Replace-funktiolla korvataan syötetystä muuttujasta pilkut pisteiksi.

Muokattavaksi muuttujaksi laitetaan Format-funktion ympäröimä Input-muuttuja. Format-funktion tarkoitus on muuttaa syötteen muoto määritellyllä tavalla. Format-funktio pystyy muuttamaan numeroita, päivämääriä ja tekstiä.

Arvot Input- ja Form -muuttujille saadaan funktion kutsusta. Input on muuttuja, jota halutaan muuttaa, ja Form on muoto, johon input muutetaan. Muoto määritellään tekstinä, jonka avulla Format-funktio parsii syötteen. Esimerkki tästä tekstistä olisi ”0.00”. Tämä muuttaisi numerosyötteen numeroksi, jossa on kaksi desimaalinumeroa näkyvillä. Toinen esimerkki muodosta olisi ”dd\.MM\.yyyy”. Tämä muuttaisi esimerkiksi päivämäärän 26. toukokuuta 2005 muotoon 26.05.2005.

DotFormatista on tehty useampi versio, eroina niiden välillä on input-muuttujan tietotyyppi. Kuvan 5 esimerkissä se on Single, mutta olemassa on myös eri versioita tietotyypeille Double, String ja Integer. Olemassa on myös versio, jossa ei ole Form-syötettä ollenkaan, jolloin funktio ainoastaan korvaa pilkut pisteillä. Käyttäjän ei tarvitse itse valita mitä versiota DotFormatista käyttää, sillä koodi pystyy valitsemaan sen syötteiden perusteella. Tätä kutsutaan metodien kuormittamiseksi, joka on ominainen toiminto .NET-kielille. (Tripathi, 2017.)

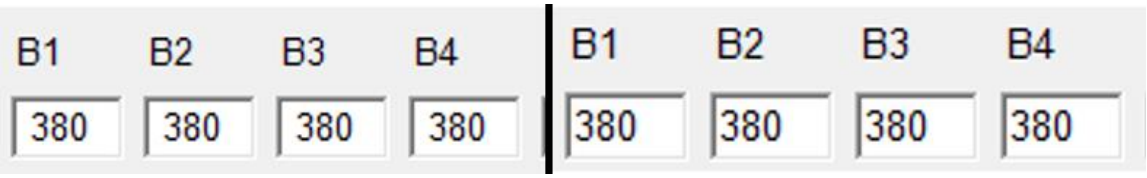
2.5.2 StringFormat -funktio

Kun numeroita muutetaan tekstiksi, käytetään siihen Visual Basicille ominaista Str-funktiota. Str-funktio muuntaa syötetyn numeromuuttujan tekstiksi. Tätä funktiota käytetään, koska se laittaa desimaalierottimeksi aina pisteen. Huono puoli tässä funktiossa on se, että se jättää numeron eteen tyhjän välin etumerkkiä varten. Tyhjä väli kertoo, että numero on positiivinen ja tyhjän välin tilalle tulee miinusmerkki, kun numero on negatiivinen. StringFormat-funktion tarkoitus on muuntaa numero tekstiksi Str-funktion avulla ja samalla poistaa tyhjä väli numeron alusta, jos se sinne ilmestyy.

```
Public Function StringFormat(ByVal input As Single)
    Return Replace(Str(input), " ", "")
End Function
```

KUVA 6. StringFormat-funktio, joka ottaa vastaan Single-tyyppisen muuttujan.

StringFormat on periaatteeltaan samankaltainen DotFormatin kanssa, minkä näkee myös kuvasta 6, eli se yhdistää kaksi funktiota yhdeksi; Replace-funktion ja Str-funktion. Replace-funktion sisällä muokattavana kohteena on Str-funktio ja sen sisällä on StringFormat-funktion parametrina syötetty Input-niminen muuttuja. Funktion tuottaman käytännön eron voi nähdä kuvassa 7.



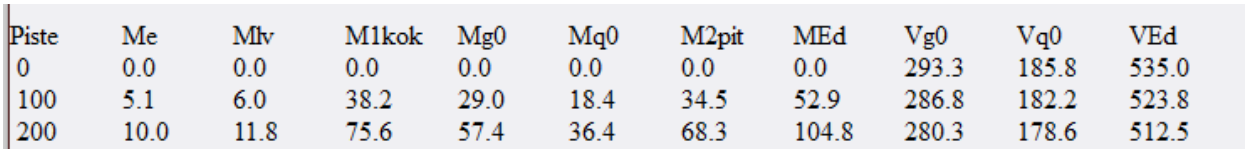
KUVA 7. Tekstikenttien ulkonäkö, korjaamaton ohjelmisto vasemmalla, korjattu oikealla.

2.5.3 Taulukot

Koska kyseessä on laskentaohjelma, on tulosten ja laskenta-arvojen taulukointi ja näyttäminen onnistuttava hyvin. Ohjelmistojen alkuperäisissä Visual Basic 6.0 -versioissa taulukot tulostettiin kirjoittamalla raakadataa suoraan käyttöliittymän taustaa vasten. Vaikka ratkaisu onkin varma ja toimiva, se vaikeuttaa sovellusikkunoiden skaalaamista ja helppoa laajennettavuutta.

Ongelman pohtimiseen käytettiin paljon aikaa ja päädyttiin siihen, että järkevintä on rakentaa taulukointi kokonaan uudelleen käyttäen TableLayoutPanel-tyyppistä elementtiä. TableLayoutPanelilla

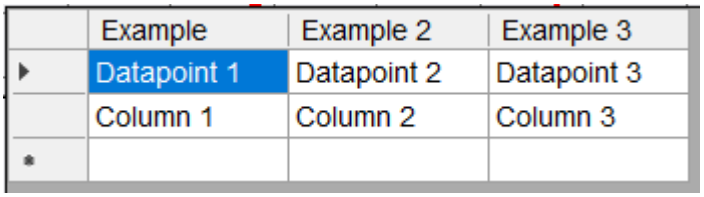
kehitys sujui hyvin ja ulkonäkö oli samankaltainen verrattuna alkuperäiseen taulukkoon, joka voidaan nähdä kuvassa 8. Myöhemmin kuitenkin huomattiin, että monimutkaisemmat taulukot skaalauksineen vaativat useita sekunteja latautuakseen, joten ratkaisu todettiin liian hitaaksi. Asian tarkempi tutkiminen paljasti vian olevan itse TableLayoutPanelissa, jota ei ole tarkoitettu suurien dynaamisten taulukkojen näyttämiseen.



Piste	Me	Mlv	M1kok	Mg0	Mq0	M2pit	MEd	Vg0	Vq0	VEd
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	293.3	185.8	535.0
100	5.1	6.0	38.2	29.0	18.4	34.5	52.9	286.8	182.2	523.8
200	10.0	11.8	75.6	57.4	36.4	68.3	104.8	280.3	178.6	512.5

KUVA 8. Muutama ensimmäisistä riveistä yhdestä alkuperäisistä taulukoista.

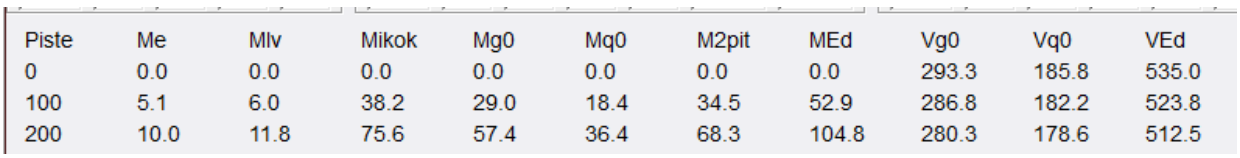
Asiaa tutkimalla selvisi myös, että sopivampi elementti saattaisi olla DataGridView, joka on tarkoitettu nimenomaan suuremmalle määrälle dynaamisesti muuttuvaa dataa. Ensimmäinen ongelma, joka tuli vastaan elementtiä lisätessä, oli elementin erilainen ulkonäkö. Uusien taulukkojen vaatimuksena on mukailla alkuperäistä ulkonäköä. Kuten kuvassa 9 näkyy, taulukon ulkonäkö on erittäin erilainen.



	Example	Example 2	Example 3
▶	Datapoint 1	Datapoint 2	Datapoint 3
	Column 1	Column 2	Column 3
*			

KUVA 9. DataGridView ilman ulkonäöllisiä muokkauksia.

DataGridView myös mahdollistaa sen sisältävän tiedon muokkaamisen, joten se oli otettava pois päältä. Seuraavaksi oli myös poistettava otsikkopalkit sekä sivulta että ylhäältä. Lisäksi oli poistettava käyttäjältä oikeudet lisätä, poistaa, siirtää, järjestää ja venyttää rivejä, sarakkeita ja soluja. Taulukon täyttävä luokka myös hakee dynaamisesti käyttöliittymän taustavärin ja asettaa sen väriksi kaikille rajoille ja taustoille, sekä poistaa ulkorajojen tyylin, jolloin päästään käytännössä alkuperäiseen ulkonäköön, joka käy ilmi kuvasta 10. Taulukkojen fontti vaihdettiin myös Arialiksi Times New Romanista, sillä sen todettiin näyttävän paremmalta ja mukailevan paremmin ohjelmiston muiden osien ulkonäköä. DataGridView toi myös mukanaan ominaisuuden maalata ja valita soluja, vaikka niitä ei voikaan muokata. Tämä mahdollistaa datan nopean kopioinnin tekstimuodossa yksittäisestä taulukosta esimerkiksi raakatekstitiedostoon tai taulukkoon Excelissä.



Piste	Me	Mlv	Mikok	Mg0	Mq0	M2pit	MEd	Vg0	Vq0	VEd
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	293.3	185.8	535.0
100	5.1	6.0	38.2	29.0	18.4	34.5	52.9	286.8	182.2	523.8
200	10.0	11.8	75.6	57.4	36.4	68.3	104.8	280.3	178.6	512.5

KUVA 10. Vertailukuva, jossa nähdään KUVA 8:n materiaali, mutta uudessa versiossa ohjelmistosta.

Taulukkojen hallinta hoidettiin täysin uudella TableClass-luokalla. Luokasta luodaan olio, johon julistaessa linkitetään taulukon elementti, jolloin luokka tietää mihin taulukkoelementtiin lisätään tietoa.

Alkuperäisessä ohjelmistossa tiedon lisäys on tehty hyvin erilaisilla tavoilla eri käyttöliittymissä, mikä vaati luokan luomiselta paljon suunnittelua. Kuormatiedot-käyttöliittymässä data on lisätty silmukalla yksi sarake kerrallaan, kun taas Mitoitustiedot-käyttöliittymän puolella data lisätään rivi kerrallaan. Tavoitteena oli saada sama luokka hoitamaan molemmat tavat.

Useamman tyylin käyttäminen tiedon lisäämiseen mahdollistettiin tekemällä lisäystä varten yksi perusfunktio, AddToTable. Sen päätarkoitus on ottaa vastaan data ja sen sijainti, jota käyttäen se sitten lisää datan oikeaan kohtaan. Oletuksena taulussa ei ole yhtään riviä tai saraketta, vaan ne lisätään dynaamisesti kaksiulotteiseen listaan, joka sitten tulostetaan lopuksi. Molempien lisäys tapahtuu AddToTablen sisällä automaattisesti. Jos kyseessä on ensimmäinen sarake kyseiselle riville, luodaan uusi lista, jonka ensimmäiseksi arvo laitetaan. Lista sitten lisätään toiseen listaan, joka sisältää jokaisen rivin omana listanaan. Jos riville on jo olemassa lista, lisätään arvo sen perään. Sarakkeen lisäys taas tapahtuu, kun ensimmäisen rivin listaan lisätään arvo. Tällöin yksinkertaisesti kasvatetaan DataGridViewin ColumnCount-muuttujaa.

Sarake kerrallaan dataa lisättäessä käytetään samoja silmukoita, kuin alkuperäisessä ohjelmistossa, mutta data lisätään AddToTablella suoraan taustaan tulostamisen sijaan. Kun kaikki data on lisätty, kutsutaan ensin ReloadTable-funktiota, josta on kaksi eri versiota. Ensimmäinen versio, jota tässä tapauksessa käytetään, ei ota vastaan leveyttä, vaan se laskee jokaiselle sarakkeelle tasaisen leveyden käyttäen sarakkeiden määrää ja taulukon kokonaisleveyttä. Toinen versio ottaa vastaan leveyden, jota sitten käytetään suoraan sarakkeiden leveytenä kuvapisteissä. Molemmat versiot tallentavat leveyden erilliseen listaan, eivät vielä suoraan taulukkoon. Lopuksi kutsutaan funktiota PrintOut, joka nimensä mukaisesti hoitaa datan näyttämisen. Printoutin voi nähdä kokonaisuudessaan kuvassa 11. PrintOut varmistaa ensin, että ReloadTable on kutsuttu ainakin kerran, jonka jälkeen se lisää listoihin tallennetun datan rivi kerrallaan, sekä kasvattaa taulukon rivien määrää asettaen samalla korkeuden lisätyille riveille. Tämän jälkeen se kutsuu erillistä funktiota, joka siirtää aiemmin tallennetut leveydet taulukkoon.

```
'Tulostaa taulun luonnin jälkeen. Tämän jälkeen on vielä mahdollista säätää kolumnien leveyttä
Public Sub PrintOut()
    If ColumnWidths.Count = 0 Then
        ReloadTable()
    End If

    For i As Integer = 0 To Labels.Count - 1
        TableInput.Rows.Add(Labels(i).ToArray)
        Dim row As DataGridViewRow = TableInput.Rows(i)
        row.Height = RowHeight * heightfactor

        For b As Integer = 0 To Labels(0).Count - 1
            Dim style As DataGridViewCellStyle = New DataGridViewCellStyle()
            style.BackColor = background
            row.Cells(b).Style = style
        Next

    Next
    SetWidths()
    TableInput.Rows(0).Selected = False

End Sub
```

KUVA 11. PrintOut-funktion implementointi.

Rivi kerrallaan datan lisääminen on kahdesta metodista käytetympi ja myös helppokäyttöisempi. Tähänkin on käytetty pohjana alkuperäisen ohjelmiston silmukoita ja lauseita, tosin paljon vanhemman Visual Basic -version tuomaa monimutkaisuutta on pudotettu pois. Itse datan lisäys tehdään kuvan 12 AddRowToTable-funktiolla, joka ottaa vastaan määrittelemättömän määrän tekstityyppisiä muuttujia, jotka sitten lisätään taulukkoon ajamalla AddToTable-funktiota silmukassa. Toinen datan lisäämistä hoitava funktio on AddToExistingRow, jolla lisätään dataa jo olemassa oleviin riveihin.

```
Table.AddRowToTable("mm", "m2", "m", "m", "m4", "m4", "m4", "m2", "m", "m", "m4", "m4", "m4", " ", "10^-3")
```

KUVA 12. AddRowToTable-funktion käyttö.

Kun kaikki data on lisätty, toimitaan hyvin samalla tavalla, kuin sarake kerrallaan lisättäessä, eli kutsutaan PrintOut. Kaiken tämän jälkeen on vielä mahdollista muuttaa taulukon kokoa kutsumalla ResizeTable, joka ottaa vastaa jokaisen sarakkeen leveyden kuvapisteinä. Leveyksiä voi määritellä vähemmän, kuin sarakkeita on olemassa, jolloin määrittelemättömät sarakkeet pysyvät oletusleveydessään. Myös leveyden jättäminen arvoon 0 säilyttää alkuperäisen arvon. Kuvassa 13 voidaan nähdä ResizeTablen käyttö taulukkoon, jossa on 15 saraketta.

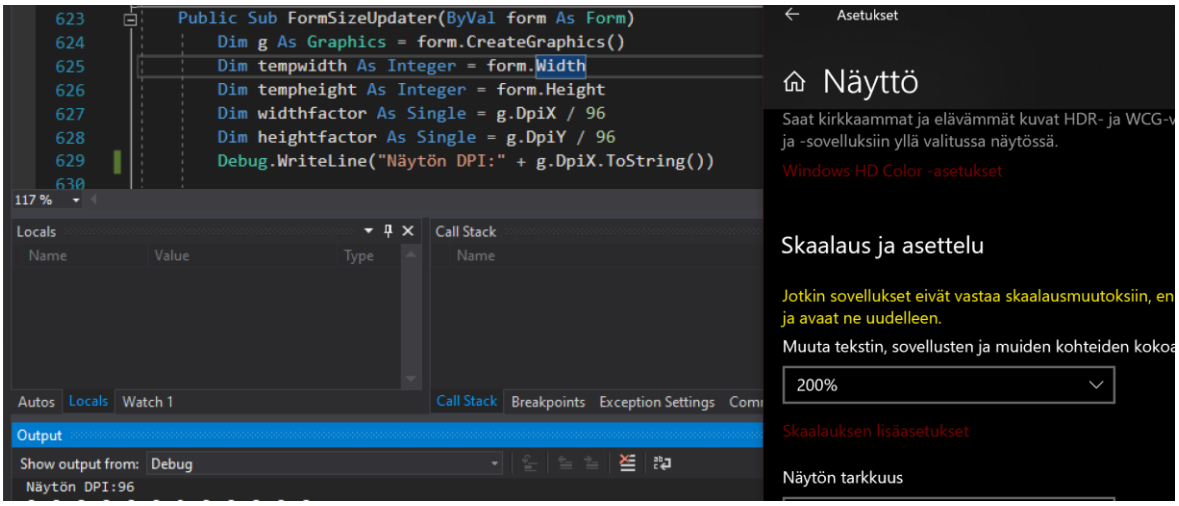
```
Table.PrintOut()
Table.ResizeTable(50, 60, 60, 60, 75, 75, 75, 60, 60, 60, 75, 75, 75, 50, 50)
```

KUVA 13. ResizeTablen käyttö.

Taulukkoihin lisättiin myös mahdollisuus sulauttaa kaksi vierekkäistä solua toisiinsa tarpeen vaatiessa. Tämä tehdään jättämällä arvoksi ””, eli tyhjä String-tyyppinen muuttuja. Jos haluaa jättää pelkän tyhjän solun, voi arvoksi antaa ” ”. Sulauttaminen itsessään tehdään lisäämällä lomakkeeseen funktio, joka aktivoituu DataGridView.CellPainting-tapahtuman yhteydessä. Funktiolla sitten manuaalisesti piirretään teksti kahden solun päälle.

2.5.4 Resoluution skaalaus

Kehityksen loppuvaiheessa ilmeni myös uusi ongelma Windowsin resoluutioskaalauksen kanssa. Windowsin versiosta 8.1 eteenpäin toimii skaalaus eri tavalla, kuin aiemmissa versioissa. Tämä ero voidaan todeta kahdesta asiasta. Ensinnäkin, jos Windows 8.1 tai uudemmalla käyttöjärjestelmällä asettaa ohjelman tulostamaan DPI (pikselitiheys) -arvon, on se aina kuvan 14 mukaisesti 96 riippumatta skaalauksen asetuksista.



KUVA 14. Näytön DPI 200% skaalauksella Windows 10:ssä on 96.

Toiseksi, uudemmissa versioissa skaalaukseen ei tarvitse tehdä mitään, vaan skaalan muuttaminen muuttaa myös ohjelmiston käyttöliittymän ja datan kokoa täysin oikein ilman mitään itse tehtyä skaalauskoodia. Ongelma ilmenee vanhemmissa versioissa Windowsista, joissa skaalaus toimi säätämällä näytön DPI-arvoa. Ongelma korjautuu useammalla muutoksella, joista ensimmäinen on vaihtaa kaikkien käyttöliittymien AutoScaleMode-arvoksi Font, jolloin teksti skaalaa automaattisesti, mutta mikään muu ei.

```
For Each cntrl As Control In form.Controls
    cntrl.Width = cntrl.Width * widthfactor
    cntrl.Height = cntrl.Height * heightfactor
    cntrl.Location = New Point(cntrl.Location.X * heightfactor, cntrl.Location.Y * widthfactor)
```

KUVA 15. Elementtien koon skaalaus.

Kaiken muun skaalaamiseen käytettiin uutta FormSizeUpdater-funktiota, joka kutsutaan jokaisen käyttöliittymän auetessa erikseen. Kun resoluution skaalaus on oletusarvossaan 100 %, on DPI-arvo aina 96, kun taas esimerkiksi kahdessasadassa prosentissa arvo on 192 (Microsoft Oy, 2017). Funktion alussa siis haetaan nykyinen DPI, joka jaetaan 96:lla. Näin saadaan kerroin, jota käytetään kokojen muuttamiseen. Aluksi muutetaan käyttöliittymän itsensä kokoa, jonka jälkeen iteroidaan dynaamisesti läpi jokainen käyttöliittymän elementti ja muutetaan niiden kokoa ja sijaintia tarpeen mukaan kuvan 15 osoittamalla tavalla. Poikkeuksia on tehtävä Microsoftin Visual Basic Power Packs -laajennuksessa mukana tulevan ShapeContainerin suhteen, jonka sisällä on LineShape ja OvalShape -tyyppisiä elementtejä. OvalShapen tietojen muokkaus hoituu hyvin samalla tavalla, kuin muidenkin

elementtien, mutta LineShape, toimii eri tavalla. LineShape edustaa piirrettävää viivaa, joka on tallennettu alku- ja loppupisteinä. Täten koon muuttamisen sijaan on skaalattava molemmat pisteet erikseen.

Skaalaus sujui pitkälti ongelmitta, yhtä epäkohtaa lukuun ottamatta. ListBox-tyyppiset elementit eivät skaalanneet sisältöään oikein, ja skaalauksen muuttaminen hävitti kaiken datan näkyvistä. Tämä korjautui muuttamalla elementin arvoa DrawMode. Oletuksena arvo on Normal, jolloin kaikki hoituu automaattisesti, mutta se voidaan asettaa arvoon OwnerDrawVariable, jolloin piirtäminen on hoidettava itse. Tämän jälkeen lisätään jokaista ListBoxia kohden funktio, joka kutsutaan DrawItem-tapahtuman yhteydessä. Tähän funktioon voidaan ottaa sitten mukaan skaalaus, kuten kuvassa 16.

```
Private Sub List1Draw(ByVal sender As Object, ByVal e As DrawItemEventArgs) Handles List1.DrawItem
    Dim myBrush As Brush = Brushes.Black
    Dim rect As Rectangle = e.Bounds
    Dim g As Graphics = Me.CreateGraphics()
    Dim gX As Single = g.DpiX / 96
    Dim gY As Single = g.DpiY / 96
    rect.Width = rect.Width * gX
    rect.Height = rect.Height * gY
    rect.Location = New Point(rect.Location.X * gX, rect.Location.Y * gY)
    e.Graphics.DrawString(List1.Items(e.Index).ToString(), e.Font, myBrush, rect, StringFormat.GenericDefault)
End Sub
```

KUVA 16. Esimerkki ListBoxin skaalaamisesta.

Osa dynaamisesti luodusta sisällöstä täytyi myös skaalata erikseen, lähinnä TableClassilla ja Drawilla tehdyt tulosteet, tosin Drawin tapauksessa ainoastaan suoraan käyttöliittymään piirrettävät ja kovakoodatut asiat vaativat skaalausta, kun taas esimerkiksi PictureBoxin leveydestä riippuvat käyrät skaalaavat automaattiesti.

2.5.4.1 Skaalaus dynaamisissa elementeissä

Ohjelmissa on useita elementtejä, joiden koko ja sijainti vaihtelevat dynaamisesti, joten niitä ei voitu kiinteästi skaalata. Nimellisesti nämä ovat piirrettävät kohteet ja taulukot. Molempien kanssa on haettu samanlaista ratkaisua, eli alussa asetetaankin kertoimet samalla kuvan 17 esittämällä tavalla, kuin muidenkin elementtien kanssa. Erona kuitenkin se, että kertoimet haetaan itse omien luokkiensa sisällä, ei käyttöliittymän koodissa.

```
Dim widthfactor As Single = g.DpiX / 96
Dim heightfactor As Single = g.DpiY / 96
```

KUVA 17. Kertoimien hakeminen piirtämistä ja taulukointia varten.

Piirrossa kertoimet on lisätty lähinnä pisteiden sijainteja määritteleviin muuttujiin, minkä on tarkoitus hoitaa skaalaus muuttamatta luokan toiminnallisuutta. Taulukoissa taas skaalausta käytetään sekä rivien korkeuden säätämiseen rivejä lisättäessä, että leveyksien muokkaamiseen, kun sarakkeiden leveyksiä määritetään tai päivitetään.

2.5.5 Ohjelman sulkeminen

Alkuperäisessä ohjelmistossa oli ongelma sulkeutumisen kanssa. Jos ohjelmaa ei lopetettu siihen tarkoitetuilla napeilla, vaan esimerkiksi ylänurkassa olevasta rastista, ohjelman kaikki käyttöliittymät sulkeutuivat jättäen kuitenkin prosessin käyntiin taustalle. Tämä ei ollut erityisen vakava ongelma korkealla prioriteetilla, sillä ohjelma ei vie erityisen paljon muistia tai suorituskykyä, joten käyntiin jääminen ei usein ollut haitallista käyttäjälle.

Ohjelmien oikein sulkeutuminen varmistettiin lisäämällä pääkäyttöliittymään lista Boolean-tyyppisiä muuttujia, joista jokainen edustaa yhtä käyttöliittymää. Arvoksi annetaan True käyttöliittymän auetessa ja False käyttöliittymän sulkeutuessa. Kun jokin käyttöliittymä suljetaan, käydään läpi kaikki arvot. Jos tällöin kaikki käyttöliittymät on merkitty suljetuiksi, ohjelma sammutetaan. Toiminnallisuudeltaan ja logiikaltaan metodi on erittäin yksinkertainen, suurempi haaste oli sen käyttöönotto, sillä se vaati jokaisen käyttöliittymän tarkkaa läpikäyntiä, jotta avaaminen ja sulkeminen päivittyvät varmasti aina. Kuvasta 18 voidaan nähdä koodi, jolla sulkeutuminen varmistetaan. Boolean-muuttujien listassa FormsOpen on kaikkien paitsi yhden muuttujan arvoksi laitettu False, sillä vain yksi käyttöliittymä aukeaa ohjelman käynnistyessä ja vaatii täten True-arvon heti ohjelman auetessa.

```
'taulukon arvot: järjestys alkaa Form1:stä, ja jatkuu numerojärjestyksessä (esim. taulukon seitsemäs arvo, eli FormsOpen(6), olisi Form7, ja FormsOpen(0) olisi Form1
Private FormsOpen() As Boolean = {False, False, False, False, False, False, True, False, False, False, False, False}

Public Sub UpdateClosed(ByVal formId As Integer)
    'päivitetään FormsOpen taulukko ja tarkistetaan pitääkö ohjelma sulkea
    FormsOpen(formId) = False
    ClosingIterator()
End Sub

Public Sub UpdateOpened(ByVal formId As Integer)
    'päivitetään FormsOpen taulukko
    FormsOpen(formId) = True
End Sub

Private Sub ClosingIterator()
    Dim Checker As Boolean = False
    Dim S As String = ""
    For i = 0 To FormsOpen.GetUpperBound(0)
        If FormsOpen(i) = True Then
            'Jos jokin form on true taulukossa FormsOpen, laitetaan checker truelle
            Checker = True
            S = S + "O, "
        Else
            S = S + "C, "
        End If
    Next
    Debug.WriteLine(S)
    'Jos checker on false, sammutetaan sovellus
    If Checker = False Then
        End
    End If
End Sub
```

KUVA 18. Funktiot, joilla käyttöliittymien avaaminen ja sulkeminen päivitetään listaan.

2.6 Uudelleenkirjoitetut funktiot ja toiminnot

2.6.1 Tulostus

Moniin toimintoihin ohjelmistoilla oli jo valmiiksi jonkinlainen ratkaisu, joka jouduttiin tarpeen vaatiessa korvaamaan modernimmalla versiolla, mutta ei kuitenkaan kokonaan alusta alkaen kirjoittamaan uudelleen, sillä toimintojen logiikka ja sisältö olivat jo olemassa alkuperäisestä koodista. Näitä uudelleenkirjoitettavia osia oli kaksi, joista molemmat olivat laajoja, mutta tulostus kuitenkin selvästi työläämpi.

Kaikkien sovellusten lasketut tulokset pystytään tallentamaan PDF-tiedostoiksi. Laajemmissa sovelluksissa pystyy tämän lisäksi suoraan tulostamaan. Näistä tiedostoista löytyvät laskutoimituksista saadut tulokset taulukoituina.

Alkuperäiset tulostustoiminnot eivät päivittyneet migraatiotapahtuman aikana, vaan muuttuivat toimimattomiksi, sillä tulostustoiminnot oli kirjoitettu VB6:lle ominaisilla tulostustoiminnoilla, kuten Printer-funktiolla ja TAB-funktiolla. Alkuperäinen tulostusfunktio luo tekstiä manuaalisella muotoilulla, eli taulukoiden paikat ja muotoilu sivulla määritellään koodissa tarkalleen. Printer-funktio lisää sivulle tekstiä, jota muotoillaan TAB-funktion ja muiden tekstimuotoilufunktioiden avulla. TAB-funktion sisälle syötetään numero, joka määrittelee tekstin kirjoituksen sijainnin tulostettavan dokumentin vaaka-akselilla.

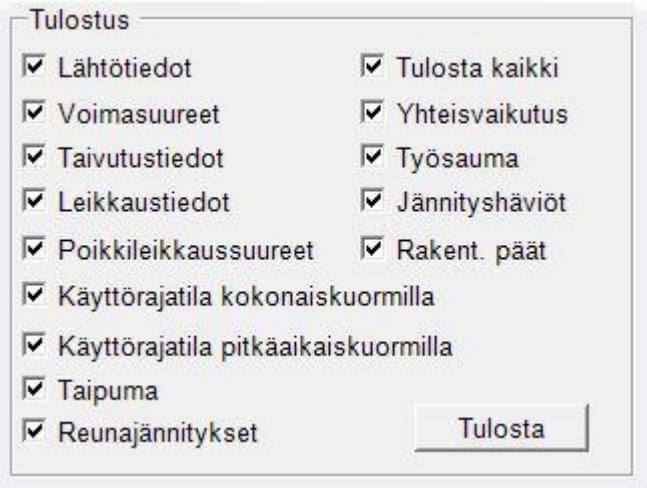
VB.NET ei sisällä näitä tulostustoimintoja, eikä niille löytynyt VB.NET:in vakiotoiminnoista sopivia korvikkeita, joten kaikki tulostusfunktiot mukautettiin käyttämään kolmannen osapuolen koodikirjastoa nimeltä MigraDoc.

2.6.1.1 MigraDoc ja Uuden tulostusfunktion rakenne

MigraDoc on Empira Software GmbH:n tuottama avoimen lähteen .NET kirjasto, jolla pystyy luomaan PDF-dokumentteja suoraan koodissa objektipohjaisesti käyttäen mitä tahansa .NET-pohjaista ohjelmointikieltä (Empira Software GmbH, 2009). MigraDoc mahdollistaa taulukkojen ja dokumentin tyylin muokkauksen ja luonnin helposti. Sovelluksissa käytetään Visual Studion NuGet-pakettipalvelun kautta ladattua PDFSharp-MigraDoc-GDI -nimistä pakettia, joka sisältää tarvittavat koodikirjastot MigraDocin käyttöön VB.NET WinForms-sovelluksessa.

Kehitykseen valittiin versio 1.32.4334 uusimman version sijasta, koska uusimmasta versiosta oli poistettu keinot dokumentin tulostusasetusten määrittelyyn ja tulostamiseen itsessään. Tulostusprosessissa käyttäjältä kysytään tulostusasetuksia, ja niiden tallentamiseen tarvitaan tätä uudesta versiosta puuttuvaa toimintoa. Samaa toimintoa käytetään myös valmiin dokumentin lähettämistä suoraan tulostimelle.

Tulostuskoodin peruslogiikka on jätetty ennalleen, joten käyttäjälle näkyvä tulostustoiminto on muuttumaton. Käyttäjä valitsee kuvassa 19 näkyvästä listasta halutut taulukot tulostukseen ja tämän lisäksi vielä tulostusasetukset (kuten fontin, fonttikoon, fontin muotoilun ja tulostuskohteen), jonka jälkeen sovellus joko tallentaa PDF-dokumentin käyttäjän valitsemaan sijaintiin, tai lähettää luodun dokumentin suoraan käyttäjän valitsemalle tulostimelle. Tulostuskoodin voi jakaa kolmeen vaiheeseen: Dokumentin luontiin, sisällön kirjoittamiseen ja tulostamiseen.



KUVA 19. Käyttäjä valitsee tulostettavat taulukot.

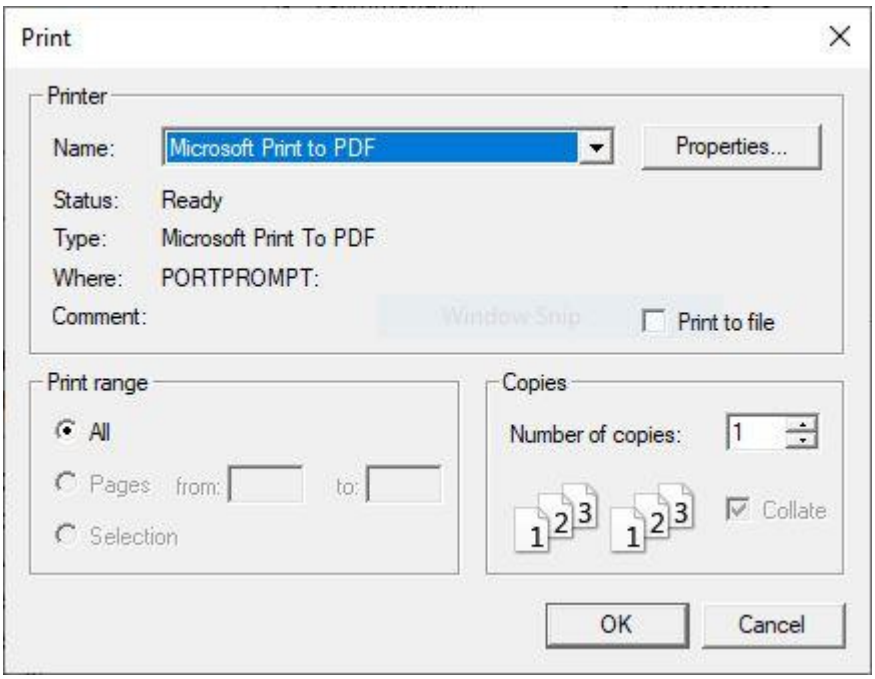
2.6.1.2 Dokumentin luonti

Dokumentin luonnissa alustetaan kaikki koodissa tarvittavat muuttujat, jonka jälkeen kysytään käyttäjältä hänen haluamansa asetukset. Oleellisimmat muuttujat ovat nimiltään Document-tyyppinen doc-muuttuja, Table-tyyppinen tb-muuttuja ja Row-tyyppinen row-muuttuja. Jokaisen sivun alussa alustetaan dokumenttiin uusi Section-tyyppinen muuttuja, joka jakaa dokumentin useaan osaan ja suorittaa sivunvaihdon. Kaikki tyypit tulevat suoraan MigraDocin kirjastosta.

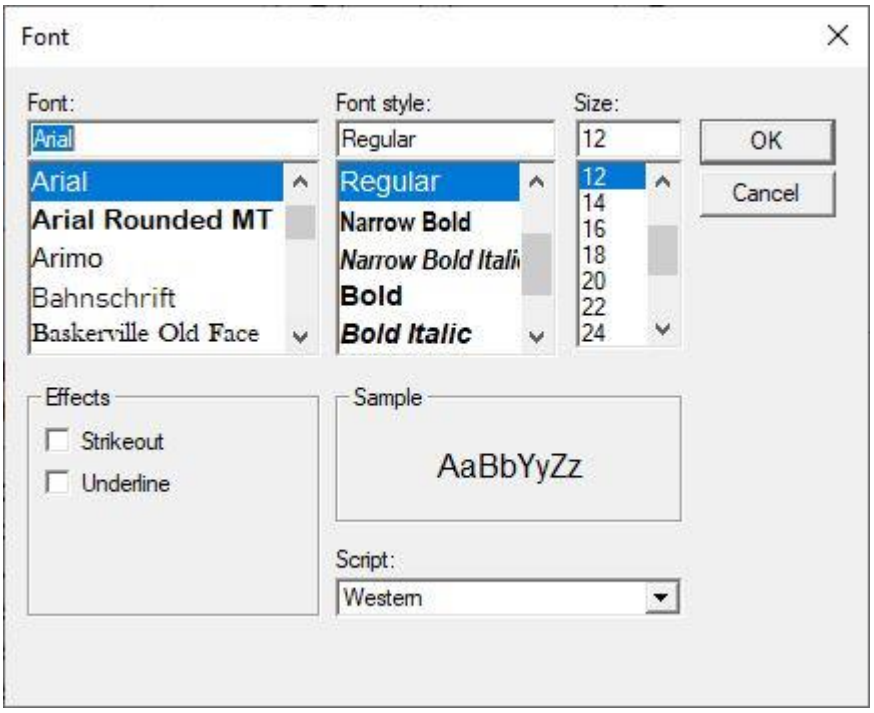
Asetuksia (kuva 21), jotka ovat käyttäjän muokattavissa, ovat:

- Fontti, joka on oletuksena Arial.
- Fonttikoko, joka on oletuksena 12.
- Fontin tyyli (kursivointi, lihavointi ja alleviivaus).
- Valinta tulostuksen ja tiedostona tallentamisen väliltä.

Riippuen valitusta asetuksesta, käyttäjä valitsee kuvan 20 mukaisesti joko tallennussijainnin tietokoneelta tai tulostimen, jolle valmis dokumentti lähetetään. Kun asetukset on valittu, tallennetaan asetukset tulostuskoodissa luotuun doc-muuttujaan.



KUVA 20. Käyttäjä valitsee, minne dokumentti lähetetään.



KUVA 21. Käyttäjä valitsee ja muokkaa fonttia.

2.6.1.3 Sisällön kirjoitus

Sisällön kirjoittaminen tehdään rivi riviltä. Jokainen uusi tekstinlisäysfunktio luo uuden rivin. Funktion sisälle voi laittaa mitä haluaa, kunhan se on tietotyypiltään string-tyyppinen.

```
doc.LastSection.AddParagraph("Syötä rivi tähän”)
doc.LastSection.AddParagraph(Format(Now, "dd\.MM\.yyyy") & " Monimutkaisempi rivi")
```

KUVA 22. Esimerkkirivien syöttäminen tulostettavaan dokumenttiin.

Kuvassa 22 on esimerkkejä tekstin syöttämisestä dokumenttiin. Doc-muuttuja vastaa itse dokumenttia. LastSection-muuttuja vastaa viimeisintä osiota, joka dokumenttiin doc on lisätty. Uuden osion lisääminen luo uuden sivun. AddParagraph-funktio lisää sen perässä olevien sulkujen sisällä olevan stringin dokumenttiin. Kuten kuvasta nähdään, ylempi rivi kuvaa yksinkertaista tekstisyötettä, joka syöttää dokumenttiin yhdelle riville tekstin ”Syötä rivi tähän”. Alempi rivi syöttää dokumenttiin tekstin ”1.5.2019 Monimutkaisempi rivi”, jossa päivämäärä on päivä, jona tulostus tehdään.

Suurin osa tulostettavien dokumenttien sisällöstä koostuu tulostaulukoista. Taulukot rakennetaan luomalla taulukkomuuttuja, johon määritellään taulukon muotoilu, tyyli ja sarakemäärä. Tätä varten rakennettiin kuvan 23 funktio CreateTable-funktio, joka luo taulukon syötettyjen arvojen perusteella. Funktion ensimmäinen syötetty arvo kertoo sarakkeiden määrän ja toinen syötetty arvo kertoo jokaisen sarakkeen leveyden senttimetreinä.

```
Private Function CreateTable(ByVal colcount As Integer, ByVal width As Double)
    'Funktion avulla alustetaan taulukkoja
    Dim tb As Table = New Table() 'luodaan taulukko
    tb.Borders.Visible = False 'Asetetaan taulukon rajat näkymättömiksi
    tb.Columns.Width = Unit.FromCentimeter(width) 'Yleinen solujen leveys. Tätä voi muokata per solu tarvittaessa
    'Luodaan tarvittavat kolumnit
    For i = 0 To colcount - 1 Step 1 'Luodaan kolumneja for-loopin avulla
        Dim column As Column = tb.AddColumn() 'Luodaan kolumni
        column.Format.Alignment = ParagraphAlignment.Left 'Ohjataan teksti vasemmalle
    Next i

    Return tb
End Function
```

KUVA 23. CreateTable-funktio.

Kuvan 23 funktio ottaa syötetyt arvot vastaan parametreina colcount ja width. Funktio alkaa tb-muuttujan alustuksella taulukkomuuttujaksi. Taulukon reunat säädetään näkymättömiksi ja width-parametri asetetaan sen leveydeksi sarakkeille, joiden mittayksikkönä käytetään senttimetrejä. Tämän jälkeen luodaan For-silmukassa sarakkeita colcount-parametrin määrän verran. Kun taulukkomuuttuja on valmis, se palautetaan takaisin ja käytetään siellä missä CreateTable-funktiota kutsuttiin.

```
tb = CreateTable(5, 1.6)

tb.Columns(3).Width = Unit.FromCentimeter(1.1)

row = tb.AddRow()
row.Cells(0).AddParagraph("Teksti 1")
row.Cells(1).AddParagraph("Teksti 2")
row.Cells(2).AddParagraph("Teksti 3")
row.Cells(3).AddParagraph("Teksti 4")
row.Cells(4).AddParagraph("Teksti 5")
For n = 1 To 19 Step 1
    row = tb.AddRow()
    row.Cells(0).AddParagraph(DotFormat(Pisteet(n), "0"))
    row.Cells(1).AddParagraph(DotFormat(Vdt(n), "0.00"))
    row.Cells(2).AddParagraph(DotFormat(Vu(n), "0.00"))
    row.Cells(3).AddParagraph(DotFormat(Asts(n), "0"))
    row.Cells(4).AddParagraph(DotFormat(Beeta1(n), "0.00"))
Next
doc.LastSection.Add(tb)
```

KUVA 24. Esimerkki taulukon luonnista tulostamista varten.

Kuvan 24 esimerkissä luodaan taulukko tb-muuttujaan käyttäen luotua apufunktiota CreateTable, joka palauttaa valmiiksi luodun taulukkomuuttujan. Toisella koodirivillä on esimerkki sarakkeen leveyden uudelleensäätämisestä. Kun taulukko luodaan, siinä säädetään kaikille sarakkeille sama leveys. Joissakin tapauksissa tiettyjen sarakkeiden täytyy olla leveämpiä tai kapeampia, jotta syötettävä teksti sopisi paremmin niihin, joten yksittäisten sarakkeiden leveyksiä säädetään erikseen jälkeenpäin.

Loppuosa kuvan 24 esimerkistä on taulukon täyttämistä riveillä. Ensimmäisenä luodaan valmiiksi määritellylle row-muuttujalle uusi rivi funktiolla AddRow. Koska taulukkomuuttujaan tb on määritelty viisi saraketta, muodostuu row-muuttujaan viiden osion pituinen taulukko. Rivin soluja voi muokata kutsumalla rivin Cells-taulukkoa. Cells-taulukon sulkujen sisällä kerrotaan numeroarvolla, mitä rivin solua muokataan. Siihen voi laittaa muita funktioita kiinni, kuten AddParagraph-funktio esimerkissä, ja syöttää tekstiä tai muuta sisältöä.

Esimerkkikuvassa 24 luodaan ensin otsikkorivi manuaalisesti ja loput rivit voidaan luoda silmukassa. Silmukan alussa luodaan aina uusi rivi row-muuttujaan ja AddParagraph-funktiota käyttäen voidaan siihen syöttää muuttujia, joiden arvo muuttuu silmukan laskurin (muuttuja ”n”) mukaisesti. Esimerkiksi silmukan ensimmäisessä syötteessä on Pisteet(n)-taulukkomuuttuja, josta valitaan tietty arvo syöttämällä indeksinumero sulkuihin, eli tässä tapauksessa numero n. Aina kun silmukka suorittaa kierroksen, kasvaa n-muuttujan arvo yhdellä, jolloin Pisteet(n)-taulukosta saadaan uusi arvo. Näitä kahta tekniikkaa käytetään suurimmassa osassa dokumentin kirjoitusprosessissa.

2.6.1.4 Vierekkäiset taulukot

Joissain tapauksissa tarvitsee dokumenttiin tulostaa kaksi taulukkoa vierekkäin. Tämä toteutetaan luomalla taulukko, jossa on kolme saraketta. Tälle taulukolle luodaan rivi, jonka molemmissa reunasoluissa on uusi taulukko.

```
Dim containerTable As Table = CreateTable(3, 1) 'luodaan uusi taulukkomuuttuja.
'Tämän avulla saadaan kaksi taulukkoa menemään vierekkäin; Taulukot syötetään tämän taulukon soluihin. Kolumnien leveyksillä ei ole väliä, ne muutetaan jälkeenpäin

'Otetaan sivun leveys talteen vähentämällä kokoleveydestä marginaalien leveydet
Dim pagewidth As Integer = CInt(doc.DefaultPageSetup.PageWidth.Centimeter) - CInt(doc.DefaultPageSetup.LeftMargin.Centimeter) - CInt(doc.DefaultPageSetup.RightMargin.Centimeter)
'Käytetään sivun leveyttä taulukon kolumnien leveyden asettamiseen
containerTable.Columns(0).Width = Unit.FromCentimeter(pagewidth * 0.58)
containerTable.Columns(1).Width = Unit.FromCentimeter(pagewidth * 0.12) 'Tämä kolumni on tyhjä kolumni, ja sen avulla erotellaan kolumnit 0 ja 2.
containerTable.Columns(2).Width = Unit.FromCentimeter(pagewidth * 0.3)
containerTable.Columns(2).Format.Alignment = ParagraphAlignment.Right
```

KUVA 25. Alustava taulukkojen luonti vierekkäisiä taulukkoja varten.

Tässä esimerkkikoodissa (kuva 25) luodaan containerTable-muuttuja, joka on normaali Table-tyyppinen taulukko. Sille asetetaan kolme saraketta. Reunasarakkeisiin asetetaan sisältötaulukot ja keskisarake jätetään tyhjäksi, jotta sisältötaulukkojen välille jää tyhjää tilaa. Seuraavaksi säädetään containerTable-muuttujan sarakkeiden leveydet. Tämä tehdään ottamalla dokumentin leveys senttimetreinä talteen pagewidth-muuttujaan. Leveydestä vähennetään samalla marginaalien leveydet, joten pagewidth-muuttujaan tallentuu kirjoitettavan alueen leveys.

Käyttäen laskettua leveyttä apuna, säädetään sarakkeiden leveydet sopivaksi. Kuvan 25 tapauksessa vasen sisältötaulukko on leveämpi, kuin oikea sisältötaulukko, joten sen sarake asetetaan leveämmäksi kertomalla pagewidth-muuttuja luvulla 0.58, kun taas oikean sarakkeen leveyden kertoimeksi asetetaan 0.3. Kun leveydet on asetettu, lisätään muualla koodissa normaalisti luodut taulukot tb1 ja tb2 sisältötaulukkoon, jonka jälkeen lisätään sisältötaulukko dokumenttiin Add-funktiolla.

2.6.1.5 Sivunvaihto

Kun yhden sivun edestä on kirjoitettu tekstiä, tulee suorittaa sivunvaihto. MigraDoc sallii kirjoittamisen ilman sivunvaihtoa, mutta teksti katkeaa sivun loppuessa ja jatkuu seuraavalla. Tämä tekee tekstistä vaikeasti luettavaa ja saattaa katkaista taulukoita, joten sivunvaihto suoritetaan luomalla osioita ja jakamalla kirjoitettu teksti ja taulukot niihin.

```
Dim sec1 As Section = doc.AddSection() 'Lisää dokumenttiin uuden osion, ja täten uuden sivun.
sec1.Tag = "Reunajännitykset" 'Asetetaan sivulle tunniste
sec1.PageSetup.TopMargin = Unit.FromCentimeter(1) 'Asetetaan sivun marginaalit
sec1.PageSetup.BottomMargin = Unit.FromCentimeter(1)
sec1.PageSetup.LeftMargin = Unit.FromCentimeter(1.5)
sec1.PageSetup.RightMargin = Unit.FromCentimeter(1.5)

'lisätään sivunumero
Dim sivunumero As Paragraph = New Paragraph()
sivunumero.AddPageField()
sivunumero.Format.Alignment = ParagraphAlignment.Right
sec1.Headers.Primary.Add(sivunumero)
```

KUVA 26. Esimerkki sivunvaihdosta.

Kuvassa 26 suoritetaan sivunvaihtoa. Siinä luodaan ensimmäiseksi Section-tyyppinen muuttuja, johon luodaan uusi osio AddSection-funktiolla (Empira Software GmbH, 2015). Seuraavaksi luotuun osiomuuttujaan määritetään asetuksia. Sille annetaan tunniste ”Reunajännitykset” ja sen marginaalit asetetaan yhden senttimetrin kokoisiksi. Kun osio on luotu, lisätään sivulle sivunumero ylätunnisteeseen luomalla uusi Paragraph-tyyppinen muuttuja nimeltään sivunumero. Tähän muuttujaan lisätään AddPageField-funktio, joka automaattisesti syöttää muuttujaan nykyisen sivun numeron. Viimeisenä sivunumero syötetään luodun osiomuuttujan ylätunnisteeseen.

Jokaiselle dokumentin sivulle tulee yleensä kaksi taulukkoa, kun käyttäjä valitsee kaikki taulukot tulostettavaksi. Tämä riippuu käytetystä laskentasovelluksesta, mutta keskimäärin tulee kaksi taulukkoa per sivu, jos sivuja tulostetaan enemmän kuin yksi. Jos käyttäjä valitsee vähemmän taulukoita, tulee sivunvaihdolla olla tarkistus sille, ettei tule liian vajaita tai tyhjiä sivuja. Tätä varten on kirjoitettu tarkistus, joka nähdään kuvassa 27.

```
If doc.LastSection.IsNull = False Then 'tarkistetaan onko dokumentissä jo olemassaolevaa osiota.
    If doc.LastSection.Tag <> "Reunajännitykset" Then 'Jos osio on olemassa, ja edellisen osion nimi ei ole ehtolauseessa oleva nimi, luodaan uusi sivu

        '(Luodaan uusi osio tässä edellisen esimerkin mukaisesti)

    Else 'Jos edellinen osio oli ehtolauseessa oleva, niin ei tehdä sivunvaihtoa, vaan tulostetaan samalle sivulle.
        doc.LastSection.AddParagraph()
    End If
Else 'Jos dokumentissa ei ole olemassaolevaa osiota, tämä toteutuu
    '(Luodaan uusi osio tässä. Tästä osiosta tulee dokumentin ensimmäinen osio)
End If
```

KUVA 27. Tarkistus, jolla estetään tyhjät sivut

Kuvan 27 tarkistus toimii siten, että ensimmäisenä se tarkistaa, että onko dokumentissa olemassa osioita. Jos ei ole, kyseinen osio on dokumentin ensimmäinen ja se luodaan normaalisti. Jos osioita on olemassa, tarkistetaan viimeisimmän osion tunniste. Jos se ei täsmää ehtolauseessa olevan tunnisteen kanssa, tuleva sisältö ei kuulu tähän osioon, joten luodaan uusi osio. Jos tunniste täsmää ehtolauseen tunnisteen kanssa, on tuleva sisältö osa tätä osiota, eli lisätään vain yksinkertainen rivinvaihto.

2.6.1.6 Dokumentin tulostaminen

Kun dokumentti on valmis, se joko lähetetään valitulle tulostimelle tai tallennetaan käyttäjän valitsemaan tiedostosijaintiin PDF-tiedostona. Aluksi koodi tarkistaa minkä vaihtoehdon käyttäjä on valinnut dokumentin luonnissa, jonka mukaan edetään.

```
If printDoc.PrinterSettings.PrintToFile = True Then
    DialResult = CommonDialog1Save.ShowDialog() 'Valitaan tallennussijainti
    If DialResult = DialogResult.Cancel Then 'Tarkistetaan jos käyttäjä painaa Cancel-nappia
        Exit Sub
    End If
    'Ensin renderöidään dokumentti
    Dim renderer As PdfDocumentRenderer = New PdfDocumentRenderer(True, PdfSharp.Pdf.PdfFontEmbedding.Always)
    renderer.Document = doc
    renderer.RenderDocument()

    'Otetaan tiedoston nimi ja polku talteen
    Dim filename As String = CommonDialog1Save.FileName
    'Luodaan tiedosto
    renderer.Save(filename)
Else
    'Renderöidään ja lähetetään tulostimelle
    printDoc.Renderer = New DocumentRenderer(doc)
    printDoc.Renderer.PrepareDocument()
    printDoc.Print()
End If
```

KUVA 28. Tulostuskohteen valinta ja tulostaminen.

Tarkistus tehdään kuvan 28 ensimmäisessä ehtolauseessa. Siinä katsotaan, onko käyttäjä valinnut tulostusikkunassa vaihtoehdon ”Print to File”, eli tulostetaanko tiedostoon. Jos se on valittu, käydään ehtolauseen If-osan koodi läpi. Jos sitä ei ole valittu, käydään ehtolauseen Else-osa läpi. Eroa näillä osilla on se, että ensimmäisessä osassa ehtolausetta avataan dialogi-ikkuna, jossa käyttäjä valitsee tiedostosijainnin, johon renderöidään ja tallennetaan luotu dokumentti PDF-tiedostona. Jälkimmäisessä osassa renderöidään ja lähetetään dokumentti printDoc-muuttujaan tallennettuun tulostimeen.

2.6.2 Piirto

Osa sovelluksista sisältää erilaisia toimintoja poikkileikkausten, käyrien ja muiden kuvioiden piirtämiseksi. Kaikki nämä toiminnot tiivistettiin yhden Draw-nimisen luokan alle. Tätä luokkaa voi käyttää joko suoraan käyttöliittymän taustaan piirtämiseen tai vaihtoehtoisesti PictureBox-objektin sisälle piirtämiseen.

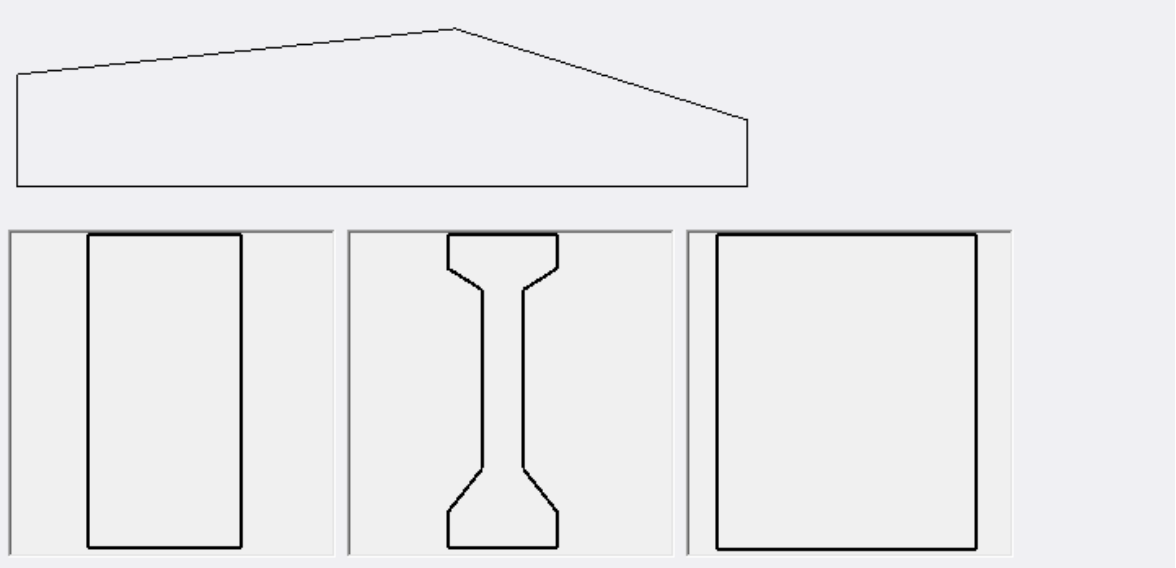
Alkuperäisissä ohjelmistoissa kuvioiden piirtämiseen käytettiin lähinnä metodia Line, jolla voitiin antaa alkupiste viivalle. Tämän jälkeen annettiin Step-komentoja, joilla otettiin ns. askelia eteenpäin. Voitiin myös määritellä, että kyseessä on laatikko, jolloin annettiin koordinaatit laatikon vasemmalle yläkulmalle ja oikealle alakulmalle.

Yksinkertaisin toiminto uudessa luokassa on viivan piirtäminen, joka saavutetaan kahdella funktiolla. Ensimmäisellä asetetaan alkupiste, jonka jälkeen toisella funktiolla lisätään uusia pisteitä ja piirretään viiva edellisestä pisteestä uuteen. Jälkimmäisestä on kaksi versiota, joista toinen ottaa vastaan muutoksen sijainnissa, eli esimerkiksi 100 kuvapistettä oikealle ja 50 alas. Toinen taas ottaa vastaan suoraan seuraavan pisteen, johon siirrytään, riippumatta edellisen pisteen sijainnista. Lisäksi on mahdollista käyttää erilaisia skaalauksia sopimaan paremmin yhteen tiettyjen arvojen kanssa. Myös yksinkertaisten nelikulmioiden piirtämiseen on kaksi funktiota, toisella voidaan tehdä muoto pelkkine reunoineen, kun taas toisella voidaan täyttää se jollain värillä. Tekstin lisäämiselle kuvioihin on myös oma funktionsa, ja piirroksen ominaisuuksia, kuten paksuutta ja väriä voi säätää lennossa. Monimut-

kaisempi toiminto on aloituspisteen vaihto, sillä normaalisti kaikki piirto alkaa vasemmasta yläkulmasta pisteestä (0, 0), jonka arvot kasvavat liikuttaessa oikealle ja alas. Toinen monimutkainen toiminto on rajojen asetus, joka on olemassa sitä varten, että voidaan rajoittaa piirtoaluetta, vaikka käytössä ei olisikaan PictureBox.

2.6.2.1 Piirron sovellukset

Piirron yksinkertaisin ominaisuus on kuvioiden piirtäminen, jota käytetään poikkileikkausten ja sivukuvien piirtämiseen lasketuista muodoista, mistä voidaan nähdä esimerkki kuvassa 29. Tämä on poikkeuksetta hoidettu asettamalla alkupiste ja siirtymällä sitten pisteestä toiseen. Pisteiden arvot on otettu suoraan vanhasta koodista, eli tämän suhteen ei ole tarvinnut tehdä muokkauksia.



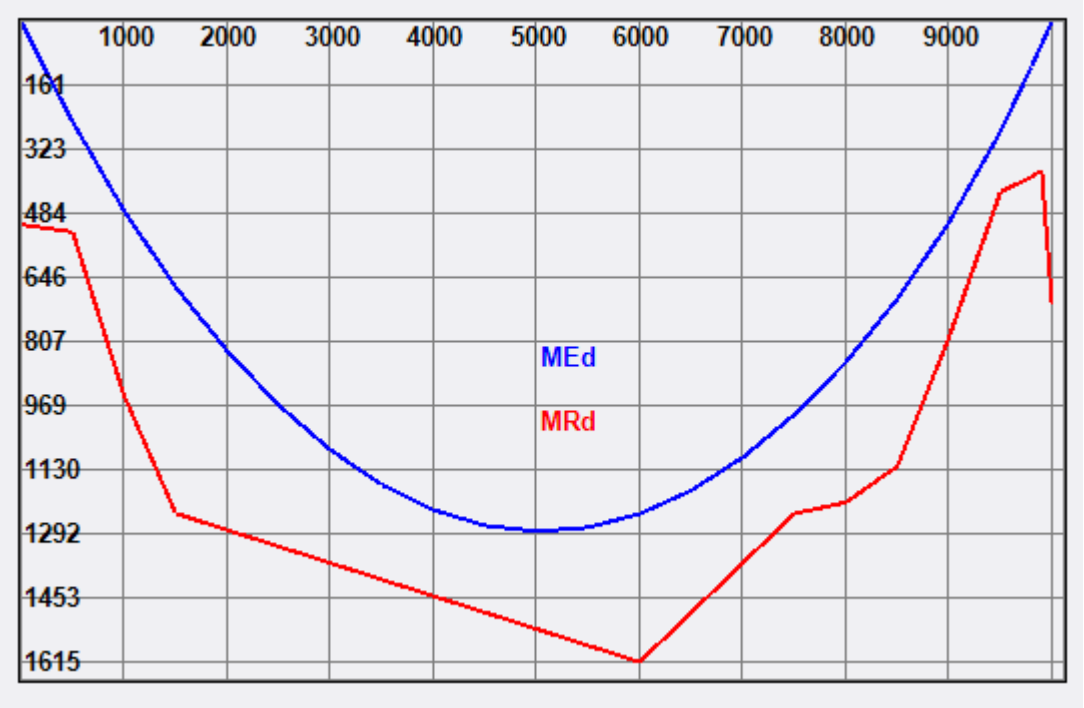
KUVA 29. Esimerkkejä piirretyistä kuvioista, ylin on piirretty suoraan käyttöliittymässä ja kolme alempaa PictureBoxeissa.

Toinen tarkoitus piirrolle on erilaisten kuvaajien ja käyrien piirtäminen ohjelman uusimmassa versiossa. Näistä ensimmäinen oli periaatteeltaan yksinkertainen käyrän piirto mitta-asteikon ja PictureBoxin avulla. Ongelmia kuitenkin koitui odottamattoman paljon, suurin kenties skaalaaminen, ei resoluution, vaan mittapisteiden mukaan. Mittapisteiden määrä ja mitattavan muodon pituus voivat vaihdella suhteellisen paljon, joten oli keksittävä tapa normalisoida ne suhteessa PictureBoxiin. Tämä tehtiin luomalla kaksi muuttujaa, xjako ja yjako, joilla kaikki suoraan lisättävät arvot jaettiin ennen piirtoa. Näiden muuttujien luonti on kuvattu kuvassa 30. Muuttuja xjako on suurimman mittapisteen ja leveyden osamäärä, kun taas yjako on suurimman saadun arvon ja korkeuden osamäärä. Korkeutta ja leveyttä on vähennetty muutaman kuvapisteen verran, jotta data mahtuu kuvaan paremmin.

```
xjako = Pisteet(ViimeinenPiste) / ReducedWidth
yjako = MaxMkap / ReducedHeight
```

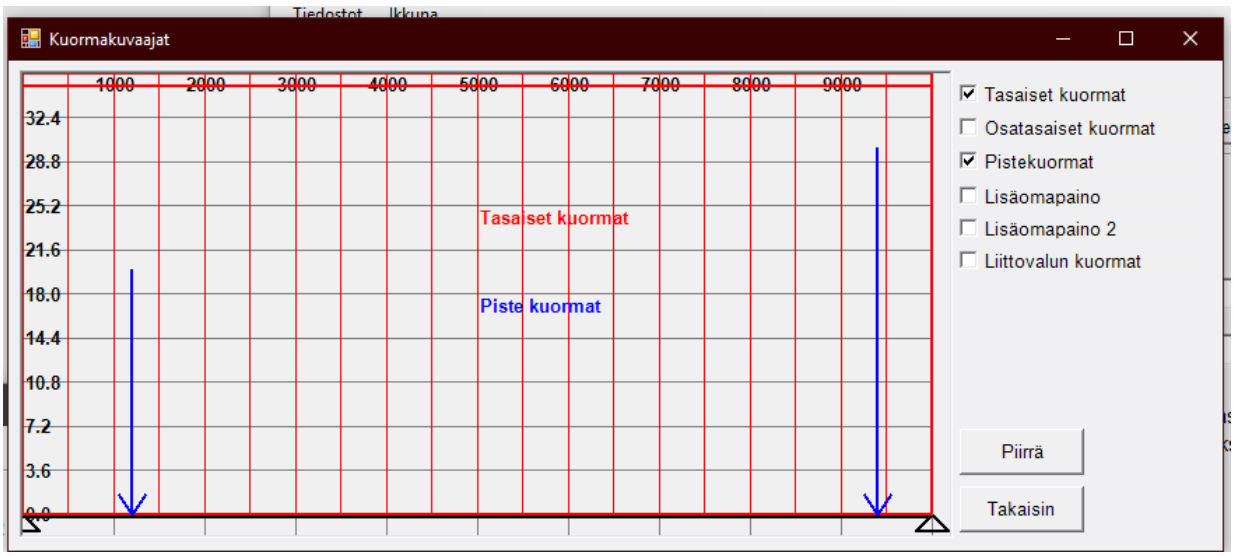
KUVA 30. Muuttujat xjako ja yjako määriteltynä.

Seuraava ongelma oli taulukon häviäminen. PictureBox, johon käyrät piirretään, on samassa käyttöliittymässä, kuin yksi DataGridViewiä käyttävistä taulukoista. Ongelma tässä on, että taulukon päivittyminen aiheuttaa PictureBoxin tyhjentymisen, jolle ei löytynyt mitään selvää syytä. Ongelma ratkaistiin hankkiutumalla eroon PictureBoxista ja siirtymällä suoraan käyttöliittymään piirtämiseen, jolloin saadaan aikaan kuvan 31 näköisiä käyriä. PictureBoxia muistuttavat rajat piirrettiin käyttäen laatikon piirtämiseen tehtyä funktiota.



KUVA 31. Käyrän piirtäminen taulukkoon. Taulukko on piirretty suoraan käyttöliittymään.

Kuvaajien piirtämistä PictureBoxien sisään ei kuitenkaan täysin hylätty, sillä ohjelmassa on käyttöliittymiä, joissa PictureBoxin tyhjentäviä päivityksiä ei tapahdu. Positiivinen puoli on se, että toisin kuin käyttöliittymään piirrettäessä, PictureBoxin kanssa ei tarvitse huolehtia rajojen ylittymisestä. Kuvassa 32 näkyvässä taulukossa on vahvasti hyödynnetty laatikon piirtämiseen käytettyjä funktioita, sillä alkuperäinen koodi käytti myös toimintoa, joka loi laatikon yksinkertaisen viivan sijaan.



KUVA 32. Kuvaajien ja muotojen piirtäminen PictureBoxin sisälle.

2.6.2.2 Käyrien piirto tarkemmin

Alun perin käyrien piirto hoidettiin niin, että käyttöliittymässä oli piilotettu nappula, jota koodissa painettiin ja näin saatiin aikaan käyrän piirtyminen. Uudempaan versioon toiminnallisuus muutettiin niin, että piirto siirrettiin oman funktionsa alle, jota voi sitten kutsua ilman simuloituja nappien painalluksia.

Käyrän sivulle ja yläosaan piirtyvät datapisteet ovat dynaamisia, eli niiden suuruus voi riippua käsiteltävästä datasta. Tämän takia on mahdotonta löytää kiinteää maksimiarvoa, jonka yli käyrät eivät nouse. Alun perin tämä oli ohjelmistoissa ratkaistu niin, että selvitettiin piirrettävien käyrien korkein mahdollinen arvo, jota käyttäen sitten skaalattiin PictureBox-elementtiä. Tämä ratkaisu ei kuitenkaan toiminut järkevästi uudemmassa .NET ympäristössä, joten skaalaus oli mietittävä uudelleen.

Skaalaukseen jätettiin käyttöön vanha korkeimman arvon etsiminen, mutta sitä käytettiin luomaan kuvissa 30 ja 35 näytetty muuttuja yjako, jota sitten käytettiin piirrettävien arvojen skaalaamiseen. Arvojen skaalaaminen todettiinkin toimivammaksi ratkaisuksi kuvan skaalaamisen sijaan ja päädyttiin siihen tulokseen, että sitä käytetään kaikissa käyrissä. Kuvassa 33 näkyvä MaxMkap on arvo, joka alustetaan aluksi nollaksi ja sitten for-silmukoiden avulla päivitetään suurimmaksi löytyväksi arvoksi.

```
'Selvittää suurimman voimasuureen
For n = 1 To ViimeinenPiste Step 1
    If MaxMkap < Mkap(n) Then
        MaxMkap = Mkap(n)
    End If
Next n

For n = 1 To ViimeinenPiste Step 1
    If MaxMkap < Voimasuuretaulukko(n, 7) Then
        MaxMkap = Voimasuuretaulukko(n, 7)
    End If
Next n
```

KUVA 33. Suurimman piirrettävän arvon etsiminen.

Käyrien selkeydelle kriittinen toiminto oli värin vaihto, joka hoidettiin yksinkertaisella funktiolla, joka nähdään kuvassa 34. Väri joudutaan vaihtamaan kahdelle erilaiselle objektille, sillä piirtämiseen käytetään molempia riippuen siitä, kirjoitetaanko tekstiä vai piirretäänkö muotoja.

```
Public Sub SetColor(ByVal color As System.Drawing.Color)
    myPen.Color = color
    brush.Color = color
End Sub
```

KUVA 34 Värin vaihto.

Apuviivat käyrille on piirretty harmaina viivoina, jotta ne eivät peitä käyriä liian vahvasti. Piirrossa on myös ominaisuus, jolla voidaan asettaa leveys suuremmaksi tietyille kuvioille, jolloin saadaan kuvien ulkonäköä säädeltyä helpommin, varsinkin käyriä varten. Kuten kuvasta 35 näkyy, on käyrien piirtäminen hoidettu myös yksinkertaisesti piirtämällä suoria viivoja pisteestä toiseen. Pisteiden runsaus kuitenkin mahdollistaa illuusion käyrän kaartuvuudesta. Kuvasta voidaan myös nähdä muuttujien yjako ja xjako käyttö käytännössä.

```
'Piirtää vaaka-apuviivat
Pic1Drawer.SetColor(System.Drawing.Color.Gray)
For n = 1 To 10 Step 1
    Pic1Drawer.CreatePoint(0, n * ReducedHeight / 10)
    Pic1Drawer.NextPoint(PictureWidth, n * ReducedHeight / 10)
Next n

'Piirtää MEd käyrän
Pic1Drawer.SetWidth(2)
Pic1Drawer.SetColor(System.Drawing.Color.Blue)
xjako = Pisteet(ViimeinenPiste) / ReducedWidth
yjako = MaxMkap / ReducedHeight
For n = 0 To (ViimeinenPiste - 1) Step 1
    Pic1Drawer.CreatePoint(Pisteet(n) / xjako, Voimasuuretaulukko(n, 7) / yjako)
    Pic1Drawer.NextPoint(Pisteet(n + 1) / xjako, Voimasuuretaulukko(n + 1, 7) / yjako)

Next n
```

KUVA 35. Apuviivojen ja yhden käyrän piirto.

2.7 Pois jätetyt toiminnot

Ohjelmistoja päivittäessä oli tarpeellista poistaa joitakin toimintoja kokonaan. Tähän oli pitkälti kaksi syytä, joista yleisempi oli se, että toimintatapa oli vanhentunut, jolloin oli yksinkertaisesti helpompi aloittaa kokonaan alusta. Toinen syy oli harvinaisempi, eli toiminnallisuuden muuttuminen. Jotkin funktiot ja lauseet olivat muuttaneet merkitystään versioiden välillä menettämättä kuitenkaan täysin toiminnallisuuttaan, joten ne saivat aikaan odottamattomia ongelmia.

Toiminnallisuuden muuttuminen sai aikaan useita ongelmia migraatioprosessin alussa, esimerkiksi ohjelman sulkeutuminen ilman selvää syytä tietyissä kohdissa tai ohjelman rakentamisen estyminen, sillä koodi oli otettu kokonaan pois käytöstä ja aiheutti virheitä kehitysympäristössä.

2.7.1 Alkuperäinen taulukointi

Taulukoinnin tapauksessa oli järkevämpää luoda kokonaan uusi systeemi taulukkojen näyttämiselle, joten vanhat metodit oli poistettava. Niistä säilytettiin ainoastaan taulukkojen näyttämisen logiikka, eli tiettyjä silmukoita ja se, milloin mitäkin näytetään.

```
CurrentY = 4000
Print "Piste"; Tab(10); "M2pit"; Tab(20); "Mr"; Tab(30); "SigmaR"; Tab(40); _
"SigmaP"; Tab(50); "SigmaS"; Tab(60); "Wk"; Tab(70); "Wkt"; Tab(80); "SigmaC"; _
Tab(90); "x"
Print "mm"; Tab(10); "kNm"; Tab(20); "kNm"; Tab(30); "N/mm2"; Tab(40); "N/mm2"; _
Tab(50); "N/mm2"; Tab(60); "mm"; Tab(70); "mm"; Tab(80); "N/mm2"; Tab(90); "mm"
```

KUVA 36. Taulukkojen alkuperäinen muoto.

Kuvassa 36 näkyvässä koodissa määritellään ensin korkeus ikkunassa muuttujalla CurrentY, mutta tämä toiminto on poistettu VB .NET -kielessä. Seuraavaksi tulostetaan otsikot kovakoodatuilla väleillä käyttäen Print-funktiota. Kyseinen funktio ei myöskään toimi uudemmassa ympäristössä, vaikka sen toiminnallisuutta voidaankin matkia tietyillä grafiikkaolioilla. Taulukkojen piirtäminen kovakoodattuna todettiin kuitenkin uudemmassa ympäristössä liian epäluotettavaksi ja vaikeasti skaalattavaksi, joten tästä ideasta luovuttiin.

2.7.2 Käyttöliittymien piilottaminen ja sulkeminen

Heti migraation alkuvaiheessa ilmeni ongelma, jonka ratkaisu oli kuitenkin yksinkertainen. Tietyt asiat saivat ohjelman sulkeutumaan ilmaan minkäänlaista ilmiselvää virhettä. Kyseessä oli automaattisen migraatioprosessin luoma virhe, joka johtui tiettyjen lauseiden merkityksen muutoksesta.

VB 6.0:ssa vanhoja lauseita oli käytetty ohjelmiston ikkunoiden piilottamiseen, mutta uusi ympäristö ymmärsi ne ikkunoiden sulkemiseksi. Ongelma ratkaistiin tilanteen mukaan joko muuttamalla lauseet piilottamaan ikkunat, tai poistamalla lauseet kokonaan.

2.8 Migraation lopputulos

Migraatiossa saatiin kaikki tärkeimmät tavoitteet suoritettua. Sovellusten siirtäminen Visual Basic 6.0-ympäristöstä Visual Basic .Net-ympäristöön onnistui, ja päivitetyt sovellukset toimivat lähes identtisesti alkuperäisiin verrattuna. Käyttäjät eivät todennäköisesti tule huomaamaan mitään suuria eroja sovellusten toiminnassa, sillä suurimmat erot löytyvät sovellusten lähdekoodista.

Koska moni tärkeä osa koodista on uudelleenrakennettu, pitää jatkokehitystä tekevien henkilöiden tutustua ja perehtyä näihin muutoksiin, ennen kuin niiden päälle voidaan jatkaa ohjelmointia. Koodi on perusteellisesti kommentoitu, joten jatkokehittäjät saavat tarvittavat tiedot lukemalla kommentteja ja tätä raporttia.

Migraatiossa oli myös lisätavoitteina kokonaan uusien toimintojen lisääminen sovelluksiin. Nämä toiminnot ovat:

- Mahdollisuus vaihtaa kieltä.
- Tapa valita ja muokata laskukertoimia.
- Yksikkötestien rakentaminen laskutoimituskoodin testaamista varten.

Nämä tavoitteet olivat kuitenkin toissijaisia verrattuna opinnäytetyössä suoritettuihin tehtäviin, ja niitä olikin tarkoitus tehdä ainoastaan, jos niille jäi aikaa.

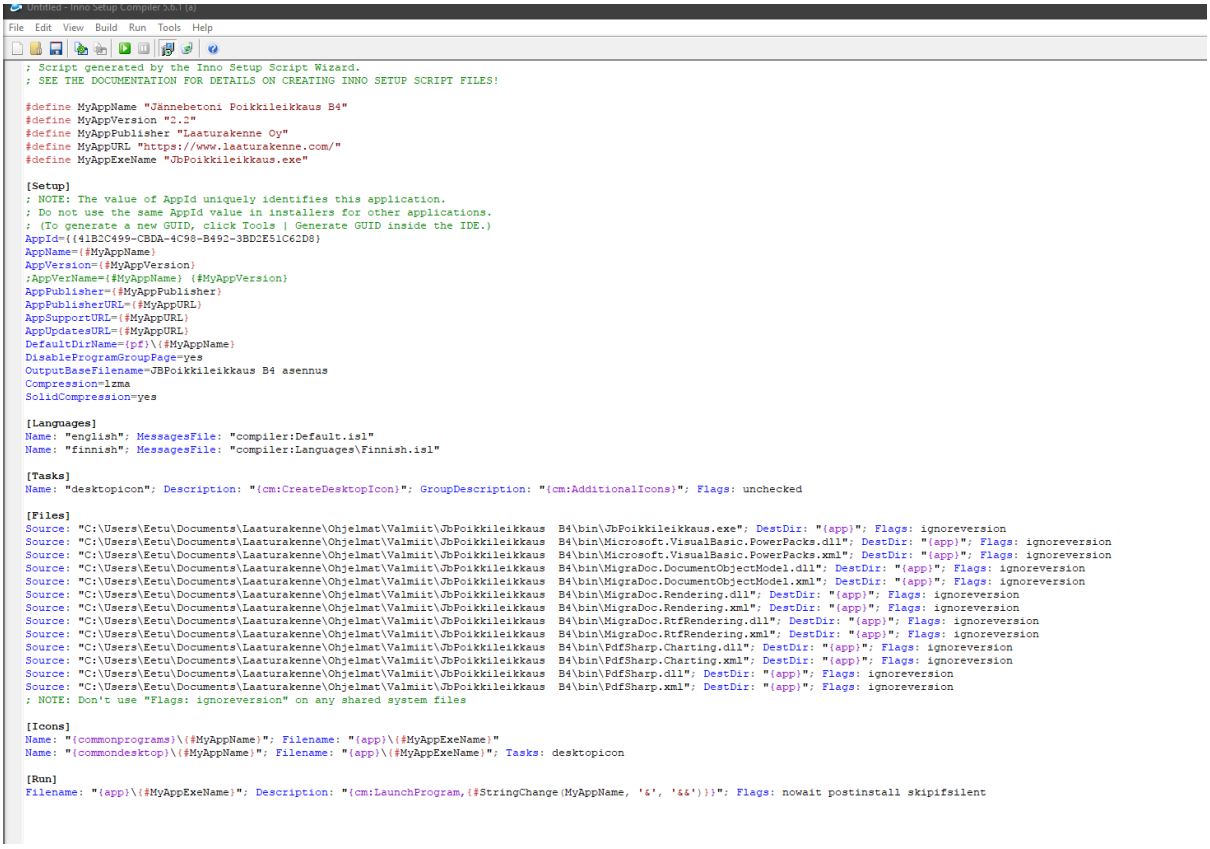
Migraation jälkeen opinnäytetyössä siirryttiin asennusohjelmien ja lisensointiominaisuuden rakentamiseen sovelluksille. Nämä oli mahdollista toteuttaa yhtä aikaa, joten Häkkinen keskittyi asennusohjelmiin ja Kovalainen puolestaan lisensointiominaisuuksiin.

3 ASENNUSOHJELMA

3.1 Inno Setup

Osa työtä oli etsiä mahdollisimman helppokäyttöinen ja joustava asennusohjelman luontiin tarkoitettu ohjelma. Tässä päädyttiin JrSoftwaren tarjoamaan Inno Setup -ohjelmistoon. Kyseessä on helppokäyttöinen ja ilmainen ohjelmisto, jolla voidaan ohjatusti luoda skripti asennusohjelman pakkaamiseen. Ohjelma sisältää valmiiksi toiminnot, joilla asennusohjelma luomiseen käytetty skripti luodaan, eikä skriptiin tarvitsisi itse lisätä mitään. (Russell & Laan, 2019.)

Tässä tapauksessa skriptiä täytyy kuitenkin muokata, ja siihen onkin lisätty itse kirjoitettua koodia, joka suoritetaan ennen asennusohjelman avaamista. Kyseinen itsekirjoitettu koodi tarkistaa tietokoneeseen asennetun .NET Frameworkin version, sillä Laaturakenteelle kehitetyt ohjelmistot vaativat ainakin version 4.6.1. Jos versio on liian vanha, annetaan käyttäjälle latauslinkki uudemman version lataamiseen.



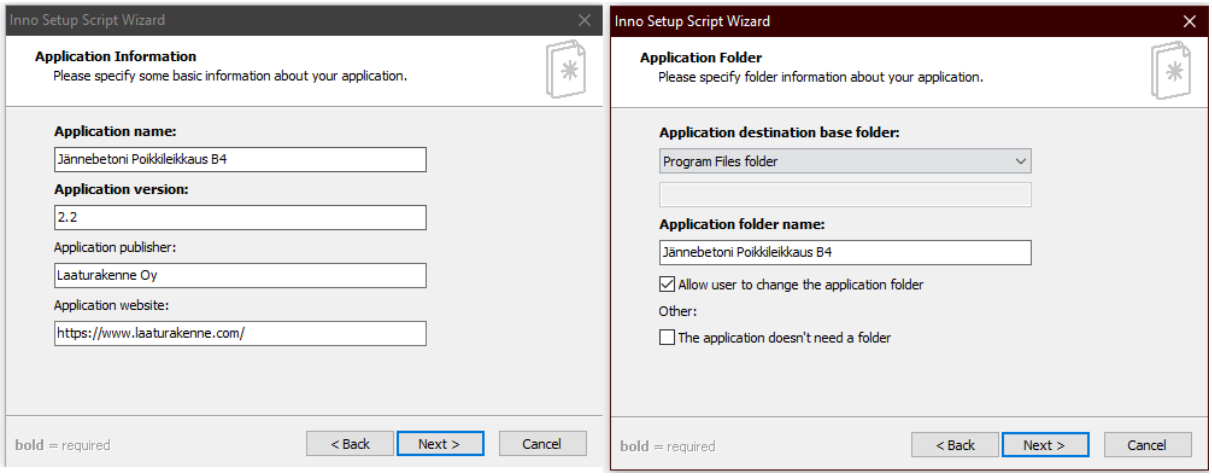
```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

#define MyAppName "Jännebetoni Poikkileikkaus B4"
#define MyAppVersion "2.2"
#define MyAppPublisher "Laaturakenne Oy"
#define MyAppURL "https://www.laaturakenne.com/"
#define MyAppExeName "JbPoikkileikkaus.exe"

[Setup]
; NOTE: The value of AppId uniquely identifies this application.
; Do not use the same AppId value in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{41B2C499-CBDA-4C98-B492-3BD2E51C62D8}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
;AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={pf}\{#MyAppName}
DisableProgramGroupPage=yes
OutputBaseFilename=JBPoikkileikkaus B4 asennus
Compression=lzma
SolidCompression=yes

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"
Name: "finnish"; MessagesFile: "compiler:Languages\Finnish.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked

[Files]
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\JbPoikkileikkaus.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\Microsoft.VisualBasic.PowerPacks.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\Microsoft.VisualBasic.PowerPacks.xml"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\MigraDoc.DocumentObjectModel.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\MigraDoc.DocumentObjectModel.xml"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\MigraDoc.Rendering.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\MigraDoc.Rendering.xml"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\MigraDoc.RtfRendering.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\MigraDoc.RtfRendering.xml"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\PdfSharp.Charting.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\PdfSharp.Charting.xml"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\PdfSharp.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\Eetu\Documents\Laaturakenne\Ohjelmat\Valmiit\JbPoikkileikkaus  B4\bin\PdfSharp.xml"; DestDir: "{app}"; Flags: ignoreversion
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
Name: "{commonprograms}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
Name: "{commondesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"; Tasks: desktopicon

[Run]
Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram,{#StringChange(MyAppName, '&', '&&')}}"; Flags: nowait postinstall skipifsilent
```

KUVA 37. Inno Setupin luomaa skriptiä.

Kuvasta 37 voi nähdä Inno Setupin luoman skriptin perusrakenteen. Skriptin alkuun on määritelty arvot muuttujille, joita on kysytty skriptiä luotaessa. Setup-osio on näiden arvojen asettamista oikeisiin paikkoihinsa, sekä asetusten säätämistä. Seuraavassa osiossa nähdään asennusohjelmalle valitut mahdolliset kielet, joita ovat tässä tapauksessa vain englanti ja suomi, sillä muut kielet eivät vielä ole tarpeellisia. Tasks-osion alla käsitellään kaikki muiden osioiden kutsumat tehtävät ja Files-osiossa listataan kaikki asennusohjelmaan pakattavat tiedostot. Seuraavaksi käydään läpi ikonit ja ajetaan

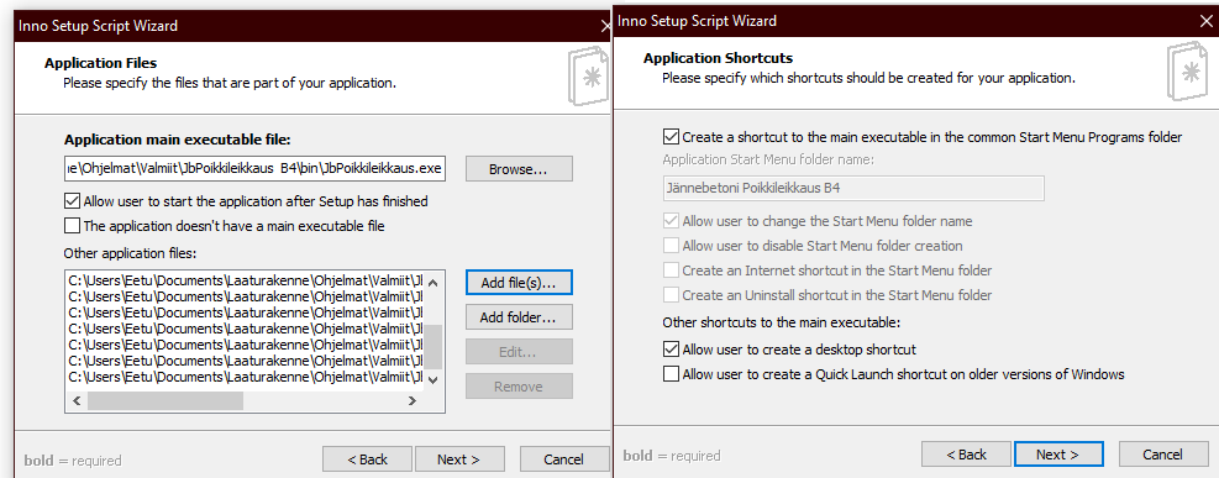
niihin liittyvä tehtävä. Lopuksi ajetaan ohjelmisto asennuksen jälkeen. Tämän jälkeen skriptiin on vielä lisätty itse luotu osio .NET Framework -version tunnistamista varten.

3.2 Asennusohjelman luonti



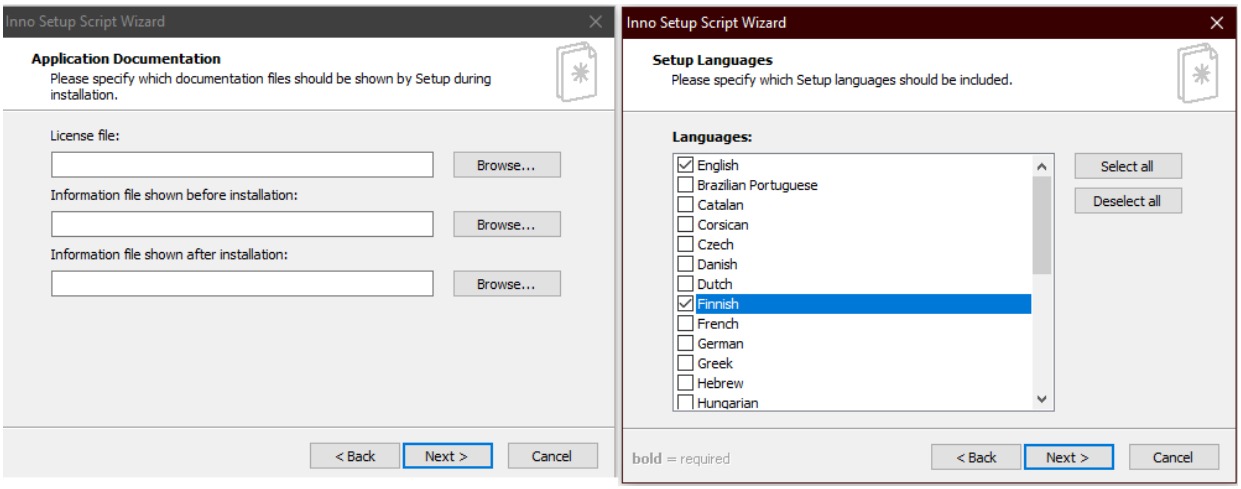
KUVA 38. Inno Setupin ohjattu asennusohjelman luonti.

Kuten kuvassa 38 näkyy, skriptin luonnissa on useita valintoja lisättävästä datasta. Ensimmäisessä ikkunassa voidaan lisätä ohjelman nimi, jota käytetään kansioiden ja linkkien luonnissa. Tämä jälkeen lisätään versio asennusohjelman metadataa varten. Lopuksi voidaan vielä lisätä julkaisijan nimi, tässä tapauksessa Laaturakenne Oy, sekä ohjelmiston nettisivujen osoite, johon on täytetty Laaturakenne Oy:n nettisivujen osoite. Seuraavassa ikkunassa voidaan valita asennuksen sijainti, joka on oletuksena Windowsin Program Files -kansio. Asennuskansion nimi on oletuksena sama, kuin ohjelmalla itsellään, mutta senkin voi tarvittaessa muuttaa. On myös mahdollista valita, että käyttäjä saa itse valita oman asennuskansion, joka on suotavaa. Toinen vaihtoehto on, että ohjelma ei tarvitse omaa kansiotaan lainkaan, mutta tämä ei ole järkevä valinta, jos käyttäjällä on itsellään vapaus valita asennussijainti.



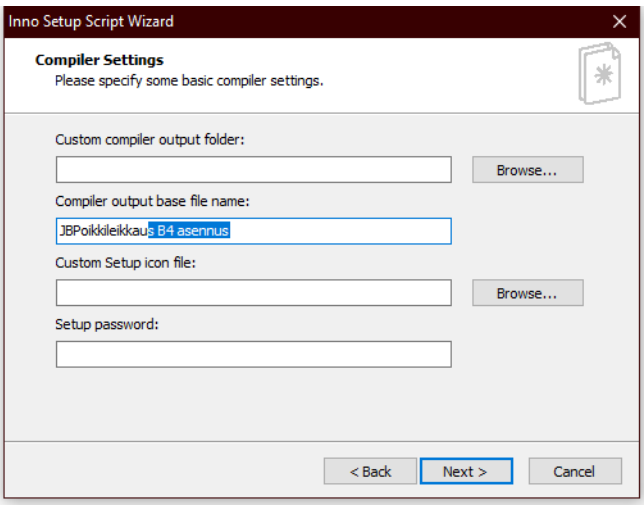
KUVA 39. Valittavat tiedostot ja linkkien asetukset.

Seuraavaksi valitaan kuvan 39 mukaisesti ohjelman vaatimat tiedostot, joita ovat exe-tiedoston lisäksi useat dll-tyyppiset ohjelmakirjastot, joita on käytetty ohjelmassa esimerkiksi tulostamisen ja lisensoinnin mahdollistamiseen. Tämän jälkeen aukeaa ikkuna, josta valitaan, miten ohjelmaa varten luodaan linkkejä. Tätä työtä varten on valittu, että käynnistysvalikkoon ei luoda erikseen omaa kansiota, vaan ohjelma lisätään suoraan valikon listaan. Jos tätä ei valittaisi, olisi mahdollista luoda oma kansio, johon voisi linkittää esimerkiksi kotisivun ja ohjelmiston poistotoiminnon. Lisäksi olisi mahdollista poistaa käynnistysvalikon kuvake kokonaan käytöstä ja antaa käyttäjälle valtuudet päättää itse kansion nimestä. Alimmat valinnat mahdollistavat työpöydän pikakuvakkeen luonnin ja kuvakkeen luonnin Windowsin Quick Launchiin, joka ei kuitenkaan ole ollut oletuksena päällä Windows 7:ssa ja sitä uudemmissa käyttöjärjestelmissä. Koska Windows 7 on vanhin ohjelmistojen tukema käyttöjärjestelmä, on ruutu jätetty valitsematta.



KUVA 40. Informaatiotiedostojen lisäys ja kielen valinta.

Tässä vaiheessa on mahdollista lisätä informaatiotiedostoja asennusta edeltämään, tai sen jälkeen, kuten kuvassa 40. On myös mahdollista lisätä erillinen lisenssitiedosto, joka näytetään käyttäjälle asennuksen aikana. Kun kaikki asennettavaan materiaaliin liittyvät valinnat on tehty, on aika siirtyä asennusohjelman asetuksiin. Seuraavaksi valitaan asennusohjelmalle kielet, joiksi tässä tapauksessa on valittu suomi ja englanti.



KUVA 41. Loppuasetusten valinta.

Skriptin luonnin viimeisessä vaiheessa, kuvattuna kuvassa 41, asetettaisiin kääntäjälle kansio, joka ei tässä tapauksessa ollut tarpeellista. Lisäksi voidaan valita luotavan asennustiedoston nimi ja ikoni, joka ei kuitenkaan ole pakollista. Salasanaa asennukselle ei ole asetettu, sillä ohjelmistojen lisenssi on suojattu muilla tavoilla. Nyt on saatu valmiiksi skripti, joka voidaan sitten ajaa asennusohjelman luontia varten.

3.3 Asennusohjelman lopputulos

Asennusohjelman suhteen päästiin asetettuun tavoitteeseen, eli löydettiin ilmainen, suhteellisen helppokäyttöinen ohjelmisto, jolla pystyy luomaan monipuolisia asennusohjelmia. Osana työtä kasattiin valmiiksi luodut asennusohjelmat palautettavaksi, sekä lisäksi kirjoitettiin ohje asennusohjelman luonnista siltä varalta, että tilaaja itse haluaa jatkaa ohjelmistojen kehitystä.

Asennusohjelmasta jäi puuttumaan ainoastaan .NET Frameworkin vaaditun version automaattinen lataus, mutta tämä ei ole vakava ongelma, sillä asennusohjelma kuitenkin varoittaa puutteesta ja antaa käyttäjälle linkin uusimman version lataamiseen.

4 LISENSOINTI

Laskentasovelluksiin tarvitaan lisensointiominaisuutta, koska ne ovat kaupallisia tuotteita ja niiden käyttö pitää suojata ja rajata niin, että vain maksavat asiakkaat voivat käyttää niitä. Ennen tämä suojaus tehtiin lukollisella USB-tikulla, joka lähetettiin asiakkaalle postitse. Tikku sisältää ohjelman ja tarvitsee avaimen, jotta sitä voi käyttää.

Sovellus toimisi vain tietokoneella, johon muistitikku on liitetty kiinni. Koska tämä on erittäin epäkäytännöllistä, toimeksiantaja tarvitsi uuden tavan suojata sovelluksensa. Ratkaisuksi mietittiin kolmannen osapuolen lisenssiohjelmaa, jonka avulla voi lisätä lisensoinnin sovelluksiin.

4.1 Lisenssiohjelma

Toimeksiantaja ehdotti lisensointisovellukseksi Soraco Techonogies-yhtiön tarjoamaa Quick License Manager-palvelua. Muitakin vaihtoehtoja tutkittiin, mutta niiden tuli täyttää vaatimukset. Vaatimukset lisensointisovellukselle olivat:

- Halpa hinta, koska sovellukset eivät ole toimeksiantajan tulojen päälähde, joten siihen ei voi käyttää liikaa rahaa.
- Suhteellisen helppo käyttö.
- Ei kuukausimaksuja.
- Pienimmät mahdolliset ylläpitovaatimukset.

Sovelluksella tuotettujen lisenssien vaatimuksina olivat:

- Aikarajallisen avaimen luonti, esimerkiksi avain, joka on voimassa vain kuukauden.
- Sovelluksen lukitseminen tietokoneelle tai mahdollisuus käyttää sovellusta millä tahansa koneella, mutta vain yhdellä koneella kerrallaan.

Muita vaihtoehtoja käytiin läpi, mutta lopulta tultiin siihen tulokseen, että Quick License Manager on paras vaihtoehto. Muut vaihtoehdot olivat joko liian kalliita, tai vaativat liian paljon työtä, resursseja tai ylläpitoa. Sellaiset vaihtoehdot oli suunniteltu suuremmille yrityksille, jotka työskentelevät pääosaisesti tietotekniikan ja lisensoinnin kanssa. Päätettiin myös, että itse ei kannata rakentaa täysin omaa lisensointitoimintoa, sillä se on tekijöiden taitojen ja kokemuksen ulkopuolella, eikä näin rakennetusta vaihtoehdosta olisi tullut tarpeeksi turvallista sovellukselle tai toimeksiantajalle.

4.2 Quick License Manager

Quick License Manager (QLM) on Soraco Techonologiesin tarjoama lisensointisovellus ja -palvelu. QLM:llä pystyy luomaan automaattisesti tarvittavat koodit lisensoinnin lisäämiseksi sovellukseen, luomaan erilaisilla vaihtoehdoilla varustettuja lisenssiavaimia. Lisäksi sillä pystyy ylläpitämään ja hallitsemaan luotuja lisenssejä ja avaimia. Quick License Managerin jokaista versiota myydään kertali-

senssillä, eli siitä tarvitsee maksaa vain kerran per lisenssi ja saadulla sovelluksella ei ole käyttörajoituksia tai kuukausimaksuja. Jos esimerkiksi haluaa QLM:n kolmelle tietokoneelle, on ostettava kolme lisenssiä. (Soraco Technologies, 2019.)

Quick License Managerista on olemassa kolme versiota. Nämä versiot ovat Express, Pro ja Enterprise. Express-versio on pienikokoisin ja halvin näistä versioista, hinnaltaan 200 € per lisenssi (Soraco Technologies, 2019). Expressissä tulee mukana vain Quick License Manager-sovellus ja sen mukana tulevat tarvittavat lisenssikoodikirjastot ja työkalut, sekä esimerkkiprojekteja niiden käytöstä. QLM Express-versiolla voi suojata ainoastaan Windows Desktop -sovelluksia, eli sovelluksia, jotka toimivat ainoastaan tietokoneilla, jotka käyttävät Windowsin käyttöjärjestelmiä. QLM Express ei automaattisesti valvo mitä lisenssiavaimia on luotu ja kenelle, vaan käyttäjän on itse kirjattava ne ylös luonnin jälkeen. QLM Express ei käytä tai tarvitse palvelinta toimiakseen, joten asiakkaan tietokoneen ei tarvitse olla yhdistettynä Internettiin tarkistaakseen lisenssin. (Soraco Technologies, 2019.)

Pro-versiossa tulee mukana Express-version sisällön lisäksi tarvittavat tiedostot ja työkalut lisenssipalvelimen luomiseen ja hallitsemiseen. Pro-version Quick License Manager sisältää ominaisuuksia lisenssipalvelimen hallitsemiseen ja uuden tavan avainten luomiseen. Sen sijaan, että avaimet luodaan ja tallennetaan manuaalisesti, QLM Pro-versio tallentaa avaimet palvelimelle tietokantaan ja lisää myös avaimen tietoja talteen, kuten kenelle avain menee, milloin se annettiin, mikä on avaimen tyyppi ja avaimen saaneen henkilön yhteystiedot. Avaimelle on myös saatavilla uusi tyyppi, tilausavain. Avaimelle voi tehdä kausitilauksen, jolle voidaan määritellä pituus ja hinta. Asiakas voi maksaa kausimaksun automaattisesti tai manuaalisesti ostamansa sovelluksen kautta. Pro-versiolla voi myös lisätä lisensointiominaisuuden useampiin sovellustyyppeihin kuin Express-versiolla. Pro-versiolla voi lisätä lisensointiominaisuuden Express-version sovellustyyppien lisäksi Windows Phone 7 ja Windows Phone 8 sovelluksiin (Soraco Technologies, 2019.)

Pro-versiolla voi myös luoda pilvipohjaisia kelluvia lisenssejä tai toistuvia lisenssejä. Kelluva lisenssi on lisenssi, joka sallii useamman käyttäjän käyttää sovellusta yhtä aikaa. Kaikilla käyttäjillä voi olla sovellus asennettuna, mutta vain tietty määrä voi käyttää niitä yhtä aikaa. Kelluva lisenssi vaatii lisenssipalvelimen toimiakseen, koska sovellus tarkistaa sieltä, ketkä ovat käyttämässä sovellusta. (10Duke, 2019.) Pro-versiolla pystyy siis hallitsemaan käyttäjille annettuja lisenssejä tehokkaammin ja yksityiskohtaisemmin, ja sen pystyy tekemään suoraan Internetin kautta.
Viimeinen versio Quick License Managerista on Enterprise-versio. Tämä versio on samanlainen Pro-version kanssa, mutta se tukee enemmän sovellustyyppejä. Nämä sovellustyypit ovat:

- Android
- iOS
- OSX
- Java
- Javascript/Electron
- Qt C++
- PHP

Enterprise-versio on tuettu myös useammassa käyttöjärjestelmässä. Aikaisemmin mainitut versiot tukevat Windows Vista/7/8/10, Windows Server 2003/2008/2012, Windows Phone 7/8 ja Windows 8/10 Store-käyttöjärjestelmiä, mutta Enterprise-versio tukee myös Android, iOS, Mac OS X ja Java Desktop -käyttöjärjestelmiä. (Soraco Technologies, 2019.)

Kaikki versiot käytiin läpi toimeksiantajan kanssa, minkä jälkeen päädyttiin siihen lopputulokseen, että Quick License Manager Express-versio otetaan käyttöön sovellusten lisensointiin. Toimeksiantajalla ei ole resursseja lisenssipalvelimen luomiseen ja ylläpitoon. Vaihtoehtona oli myös lisenssipalvelimen isännöinti Soracon tarjoamalla palvelimella vuosimaksua kohtaan, mutta se olisi tullut liian kalliiksi. QLM Express ostettiin kertamaksulla ja lisensoinnit rakennettiin sovellukseen sen avulla. Tämä tarkoittaa sitä, että toimeksiantajan on itse pidettävä huoli kaikista lisenssiavaimista ja niiden tiedoista, kuten vastaanottajista, ostopäivistä, lisenssiavaimien sulkemispäivistä ja luontiasetuksista.

4.2.1 Käyttöönotto

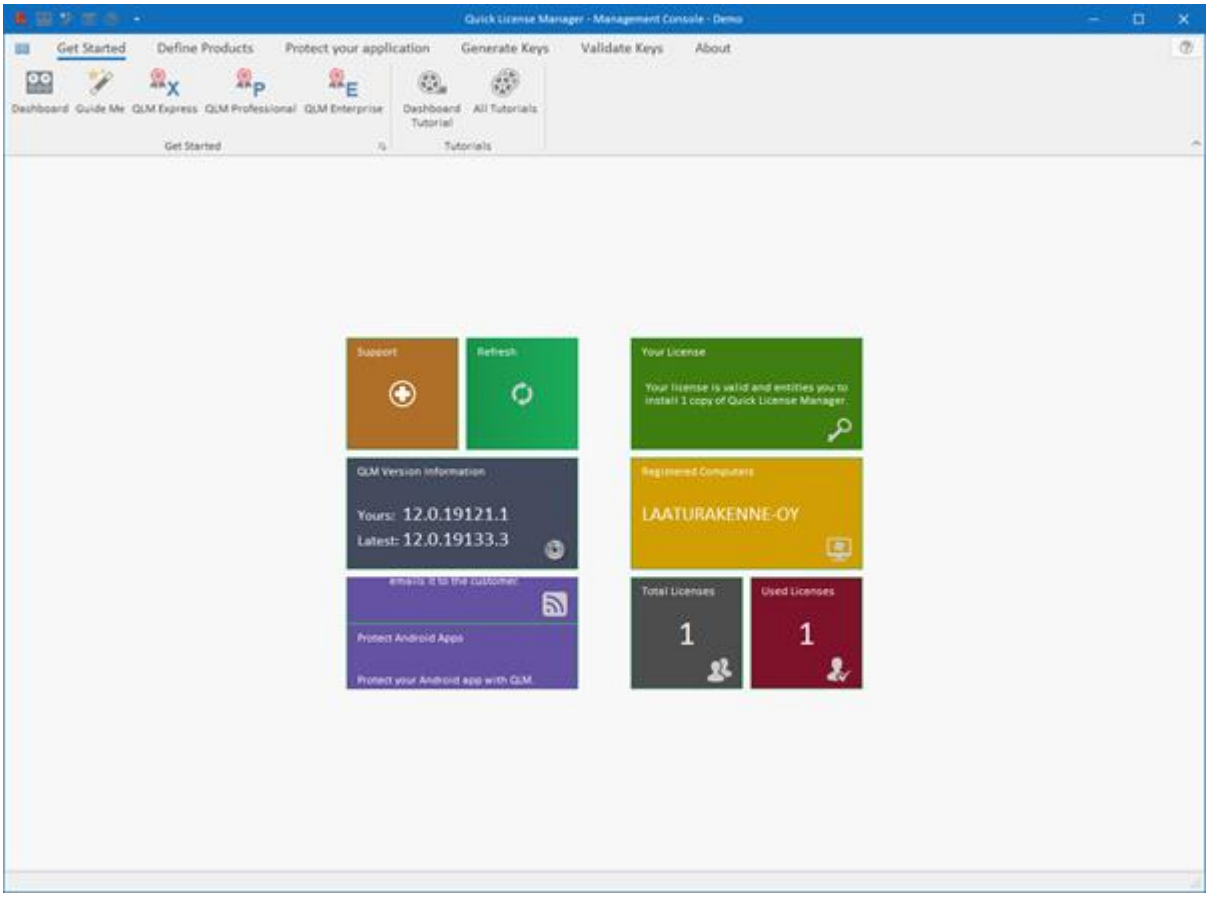
Quick License Manager Express-version käyttöönotto on hyvin yksinkertaista. Kun sovelluksen asennuspaketti on ladattu, ei tarvitse kuin kaksoisklikata sitä ja kulkea ohjattu asennusprosessi läpi, kuten normaalisti sovelluksia asennettaessa. Kun asennusprosessi on suoritettu, on Quick License Manager käyttövalmis.

Jotta Express-version lisensointi toimii, tulee lisenssin tekijän ja asiakkaan koneella olla asennettuna Microsoftin .NET Framework versio 4.5.2 tai suurempi. Laskentasovelluksilla on vaatimuksena .NET Framework 4.6.1 tai suurempi, joten kun sen asentaa, täytetään samalla vaatimukset lisensoinnin toimimiselle. Jotta lisensointiominaisuudet voitiin lisätä laskentasovelluksiin, täytyi ensin määritellä laskentasovellukset tuotteiksi Quick License Managerissa ja sitä kautta generoida tarvittavat koodit ja tiedot lisensoinnin lisäämiseen.

4.2.2 Hallintasovellus ja aloittaminen

Quick License Manager Express-version mukana on vain yksi sovellus ja se on nimeltään Quick License Manager-hallintasovellus. Hallintasovelluksessa on kuusi välilehteä, jotka ovat nimiltään Aloitussivu (Get Started), Määrittele tuotteet (Define Products), Suojaa sovelluksesi (Protect Your Application), Tuota avaimia (Generate Keys), Vahvista avaimia (Validate Keys) ja Tietoja (About).

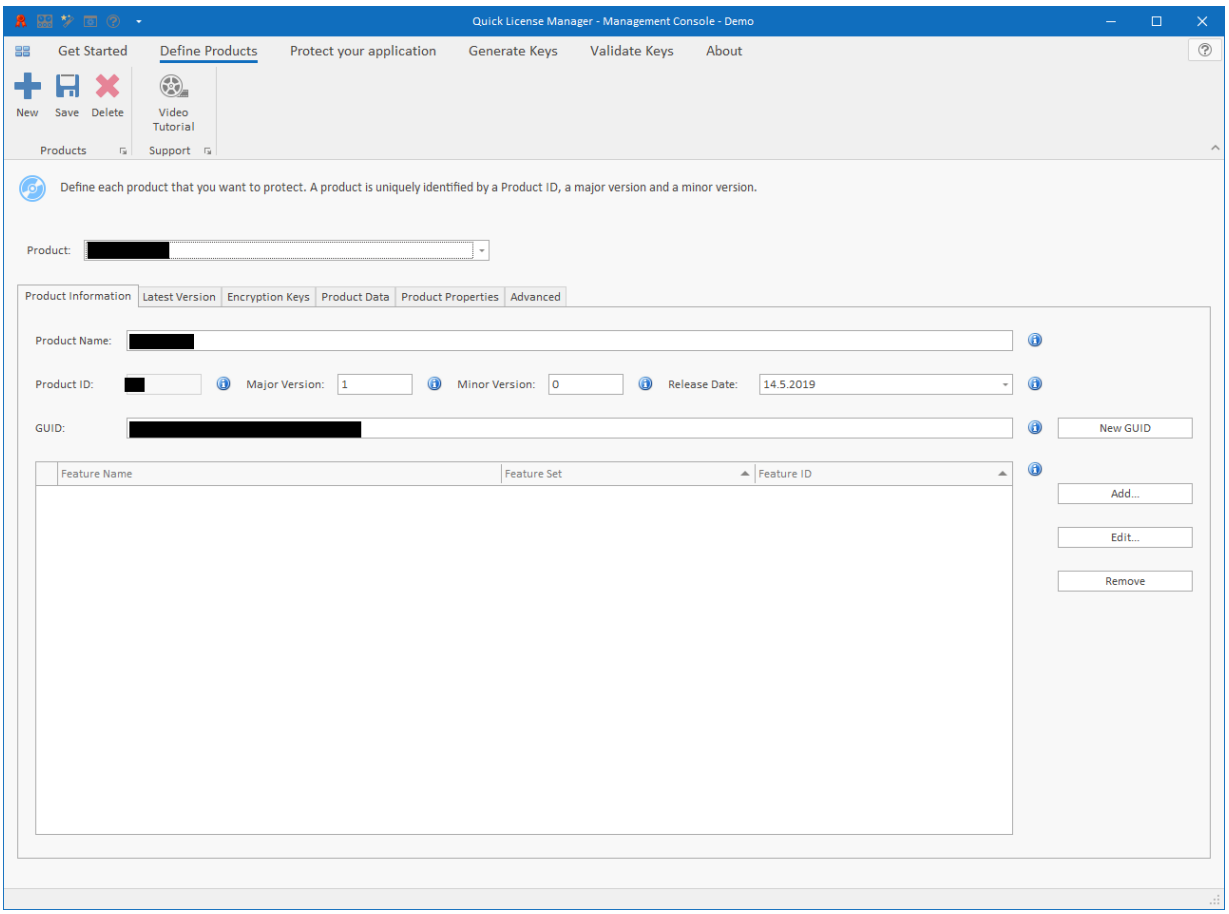
Aloitussivulta, joka nähdään kuvassa 42, löytyvät sovelluksen kotisivu ja kaikki perustiedot ensisilmäyksellä. Esimerkkejä sieltä löytyvistä tiedoista ovat sovelluksen versio, kenelle se on lisensoitu ja kuinka monta tuotetta on määritelty. Sieltä löytyy myös oppaita sovelluksen käytölle. Oppaat ovat joko sovelluksen sisäisiä tekstioppaita tai pikalinkkejä QLM:n esimerkkiprojekteihin ja Youtube-nettisivustolta löytyviin opasvideoihin.



KUVA 42. Quick License Managerin aloitussivu.

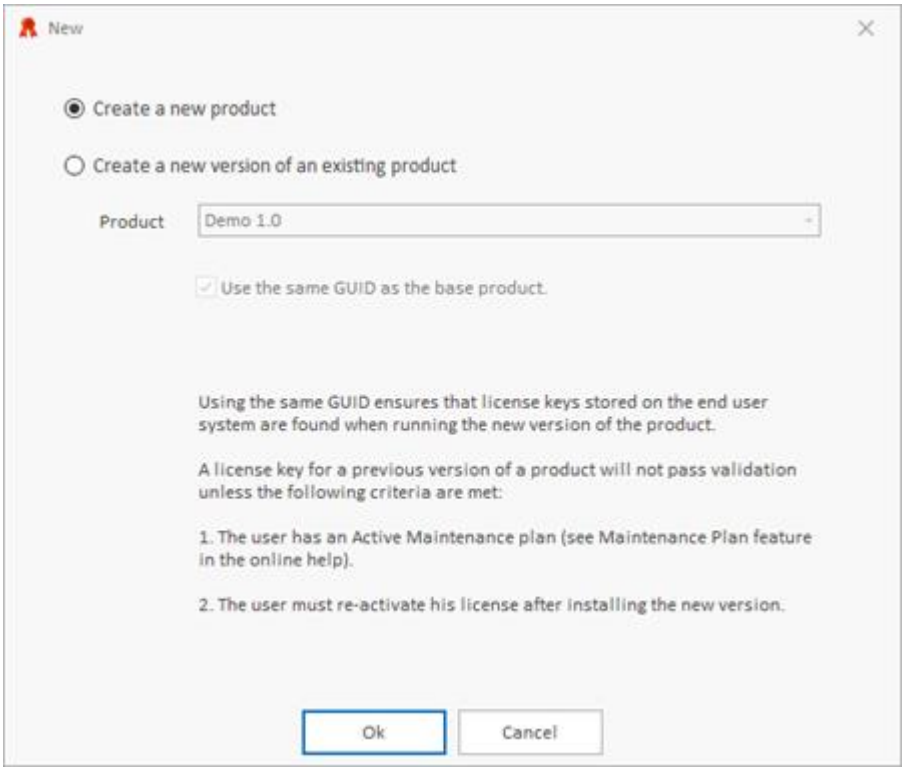
4.2.3 Tuotteen määrittely

Määrittele tuotteet -sivulla luodaan tuotetiedot, joiden avulla suojataan laskentasovellukset ja luodaan niille uniikit avaimet.



KUVA 43. Hallintasovelluksen Määrittele tuotteet-ikkuna.

Kuvan 43 sivulla on mahdollista luoda jokaisesta laskentasovelluksesta oma tuote, johon on tallennettu sovelluksen tiedot, joiden perusteella voidaan sovellukselle luoda lisenssiavaimet. Tuote luodaan painamalla ikkunan vasemmasta yläkulmasta New-nappia. Tämä avaa kuvan 44 dialogi-ikkunan, jossa ohjelma kysyy, että halutaanko luoda uusi tuote, vai halutaanko muokata vanhaa tuotetta.



KUVA 44. Uuden tuotteen luonti.

Kun uusi tuote on luotu, ilmestyy tuotelistaan uusi tuote nimellä ”New Product 1.0”. Seuraavaksi on muokattava luodun tuotteen tiedot. Tärkeimmät tiedot, joita on muokattava, ovat:

- ”Product Name”, eli tuotteen nimi.
- ”Major version” ja ”Minor version”, eli tuotteen versionumero.
- ”Release Date”, eli julkaisupäivämäärä.

Sivulta löydyy myös GUID, joka toimii tuotteen yksilöllisenä tunnisteena. Tämä luodaan automaattisesti, joten sitä ei tarvitse muokata.

Tämä sivu sisältää myös välilehtiä, joista tuotetta voi muokata. Välilehdet Latest Version, Product Data ja Product Properties toimivat vain Pro-versiossa ja ylöspäin, joten niitä ei käytetty laskentasovellusten lisensoinnin luonnissa. Tarvittavat tiedot lisensoinnille voidaan löytää välilehdistä Product Information ja Encryption Keys. Jälkimmäisestä välilehdestä löytyy julkinen salausavain, jota lisensoitava sovellus käyttää apuna lisenssiavainten tarkistamisessa. Kun tuotteen tiedot on asetettu, painetaan Save-painiketta QLM:n ikkunan vasemmasta yläkulmasta.

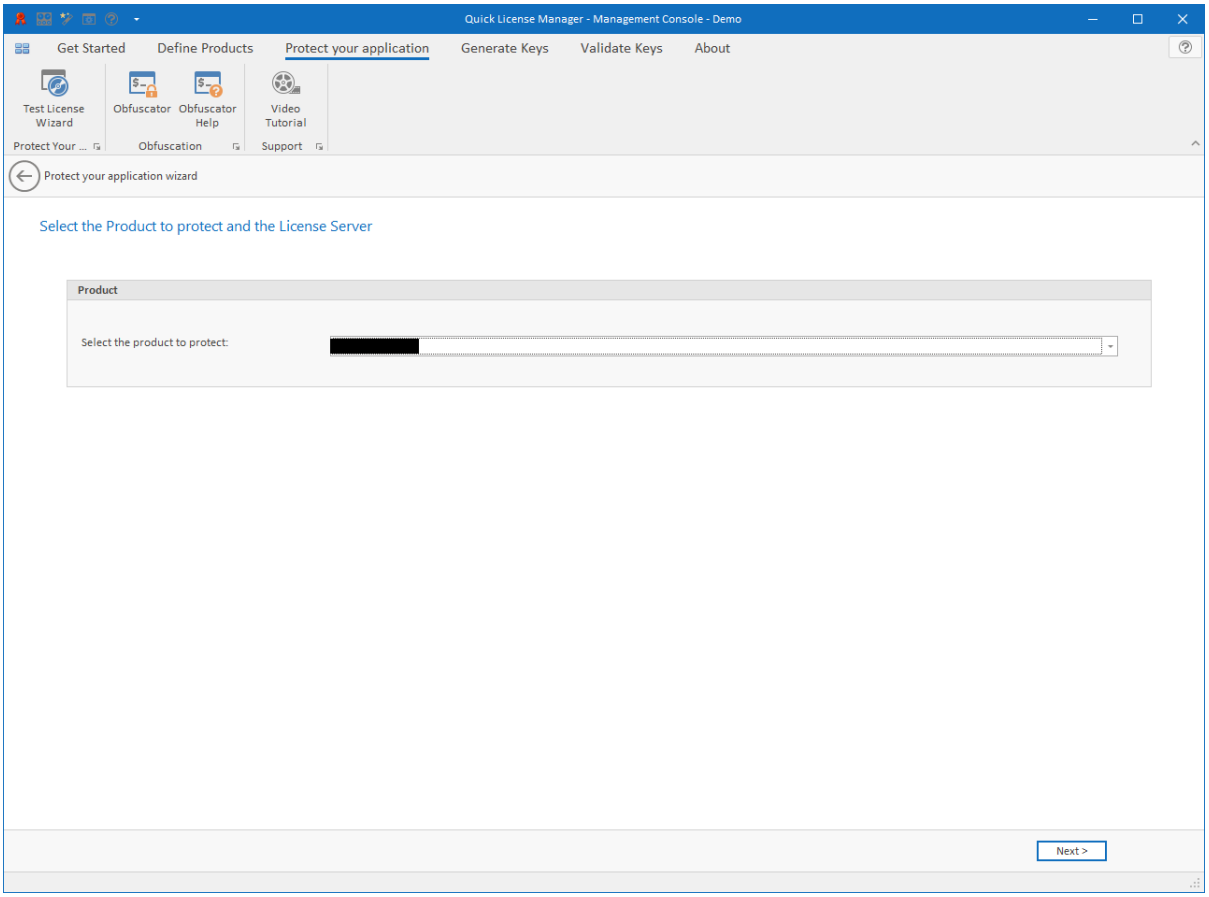
Tällä tavalla luotiin tuotetiedot kaikille laskentasovelluksille. Kun tuotteet olivat valmiita, siirryttiin Suojaa sovelluksesi-sivulle luomaan automaattiset koodilisäykset sovellusten lähdekoodeihin.

4.2.4 Sovelluksen suojaus

Tällä sivulla QLM:n hallintasovellus kysyy tuotteen, jolle luodaan lisensointikooditiedostot. Tämän jälkeen sovellus kysyy tietoja lisensoitavasta ohjelmasta. Kun välilehdelle ensimmäisen kerran

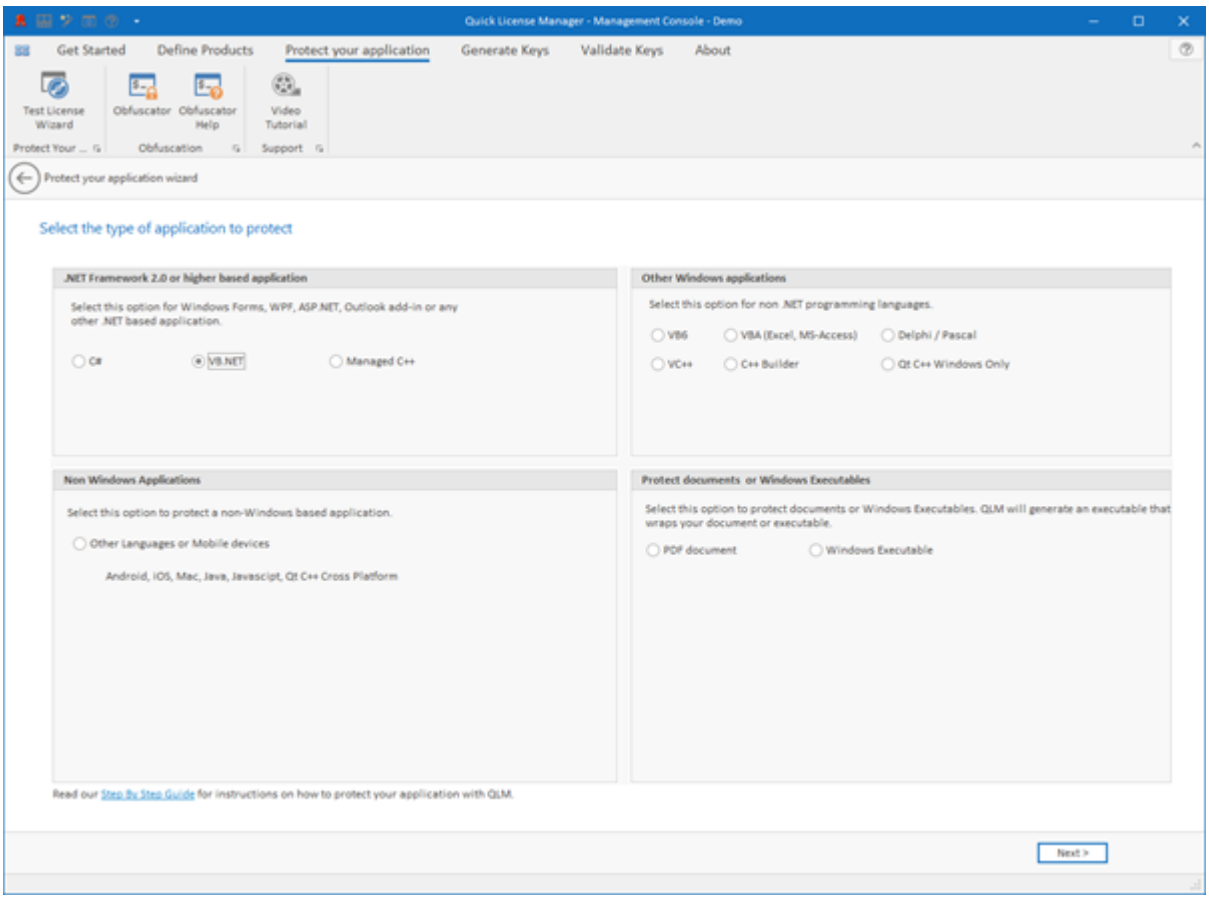
siirrytään, siellä kerrotaan suojausprosessin askeleet ja ohjeita siihen. Sieltä aloitetaan suojausprosessi Seuraava-nappia painamalla.

Suojausprosessin ensimmäinen askel on suojattavan tuotteen valitseminen kuvan 45 pudotusvalikosta. Jos käytössä olisi ollut Pro- tai Enterprise-versio, tällä sivulla olisi myös kysytty, että mitä lisenssipalvelinta käytetään tälle tuotteelle.



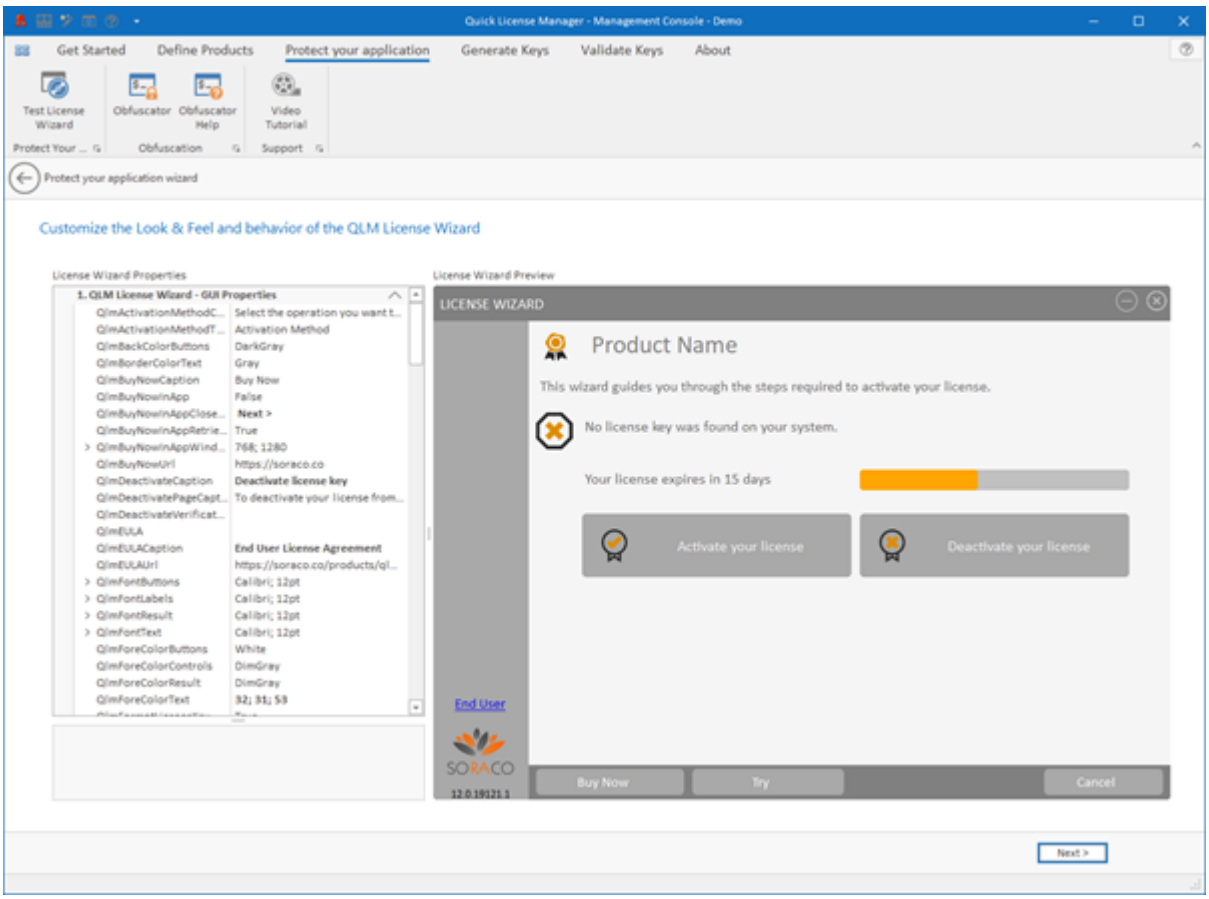
KUVA 45. Suojaa sovelluksesi-sivun ensimmäinen vaihe: tuotteen valitseminen.

Tuotteen valinnan jälkeen avautuu kuvan 46 sivu, jossa valitaan suojattavan sovelluksen tyyppi. Tyypillä tarkoitetaan ohjelmointikieltä, jolla sovellus on rakennettu ja minkä tyyppinen sovellus on kyseessä. Laskentasovellukset ovat Visual Basic .Netillä rakennettuja Windows Forms-sovelluksia, joten ruudusta valittiin ”.NET Frameworks 2.0 or higher based applications” -vaihtoehdon alta VB.NET.



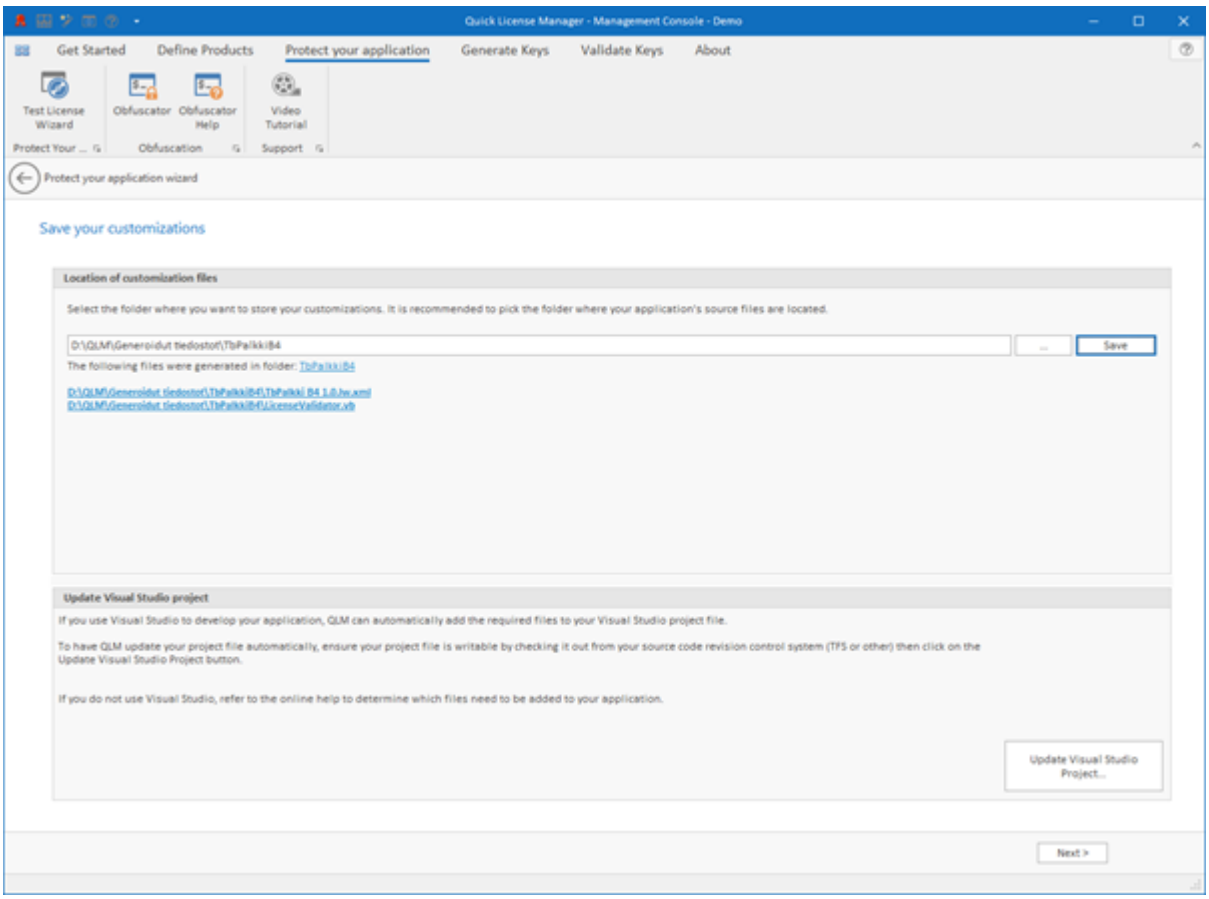
KUVA 46. Suojaa sovelluksesi-sivun toinen vaihe: suojattavan sovelluksen tyyppi.

Sovelluksen tyypin valinnan jälkeen avautuu sivu kuvasta 47, jossa luodaan lisensointikoodin mukana tuleva lisenssiavustaja. Lisenssiavustaja on käyttäjälle näkyvä sovellus, jonka avulla käyttäjä voi syöttää lisenssiavaimensa tarkistettavaksi. Sivulla voidaan valita ja muokata monia lisenssiavustajan toimintoja, kuten esimerkiksi näkyviä painikkeita, lisenssin tarkistustapaa, tai ulkonäöllisiä asioita, kuten sovelluksen värejä ja fontteja.



KUVA 47. Suojaa sovelluksesi-sivun kolmas vaihe: lisensointiavustajan muokkaus.

Päätettiin, että laskentasovelluksiin rakennetaan sisälle oma lisensointi-ikkuna omana käyttöliittymänään, joten lisensointiavustajan muokkaus ohitettiin siirtymällä suoraan viimeiselle sivulle. Seuraavaksi siirrytään kuvan 48 sivulle, jolta valitaan tallennussijainti luotaville kooditiedostoille. Kooditiedostot ovat nimeltään LicenseValidator.vb ja tuotteen mukaan nimetty xml-tiedosto, joka sisältää asetuksia. Ne voidaan tallentaa haluttuun sijaintiin, tai suoraan lähdekoodikansioon. Jos ne tallennetaan lähdekoodikansioon, pystyy QLM hallintasovellus automaattisesti lisäämään luodut tiedostot ja tarvittavat viittaukset projektiin Visual Studio-integraation avulla.

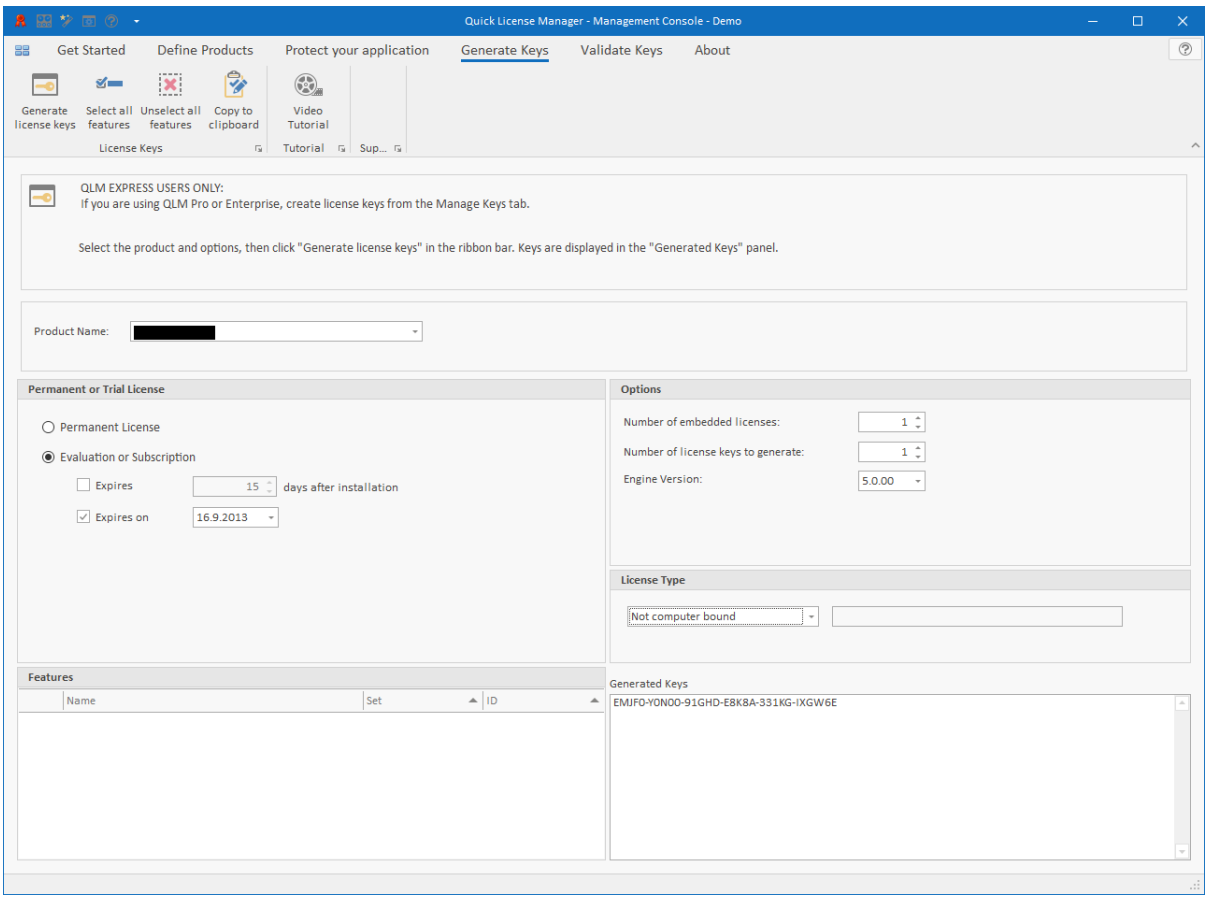


KUVA 48. Suojaa sovelluksesi-sivun neljäs vaihe: luotujen tiedostojen tallennus.

Valitettavasti Quick License Managerin generoimat tiedostot, varsinkin LicenseValidator.vb, luotiin yleiskäyttömielessä, eli ne sisältävät funktioita ja koodia palvelimen kanssa kommunikointiin, ja toimintoja ja vaatimuksia, jotka toimivat vain Pro- tai Enterprise-versioissa. Tämä korjattiin etsimällä QLM:n mukana tulleista esimerkkiprojekteista Expressille rakennettu VB.NET projekti ja ottamalla sieltä pohjaksi LicenseValidator.vb. Sieltä löydettiin hyvässä muodossa kaikki tarvittavat toiminnot, tuotteen tiedot jouduttiin ainoastaan muokkaamaan oikeiksi, minkä jälkeen muokattiin ja lisättiin vielä muutamia rivejä koodia.

4.2.5 Avainten luonti

Kun tuotteet on luotu, lisenssikoodit generoitu ja lisätty sovelluksiin ja kaikki muut lisäykset ja korjaukset tehty, voidaan aloittaa lisenssiavaimien luominen. Tämä tehdään Tuota avaimia-sivulla.



KUVA 49. Tuota avaimia-sivu.

Kuvassa 49 näkyy valinta tuotteelle, jolle avain luodaan, mitä asetuksia voi valita avaimelle, ja esimerkkiavain oikeassa alanurkassa olevassa tekstiruudussa. Express-versiossa voi luoda vain pysyviä avaimia, joiden aika ei koskaan mene umpeen ja kokeilu- tai tilausavaimia, jotka ovat väliaikaisia. Jos valitsee kokeilu- tai tilausavaimen, sille voi säätää päivämäärän tai ajan, jolloin se menee umpeen. Laskentasovelluksia varten tehdään tilausavaimia, joille säädetään tietty päivämäärä, jolloin ne umpeutuvat. Oikealta puolelta sivua voidaan valita avainten luonnille asetuksia, kuten kuinka monta lisenssiä laitetaan avaimeen mukaan, kuinka monta avainta luodaan generointia kohden ja mitä moottoriversiota käytetään. Yhteen avaimeen voi laittaa useamman lisenssin, eli sitä avainta voi käyttää monta kertaa. Tämä toiminto toimii vain Pro- tai Enterprise-versioissa, sillä tämä toiminto tarvitsee palvelimen, jota vasten pitää tarkistaa kuinka monta kertaa avainta on käytetty. Moottoriversiolla tarkoitetaan avainten generointimoottoria. Siitä on useampi versio, joten käyttäjä voi valita käyttöön vanhemman moottorin, jos tällä on vanhempi versio lisensointikoodista.

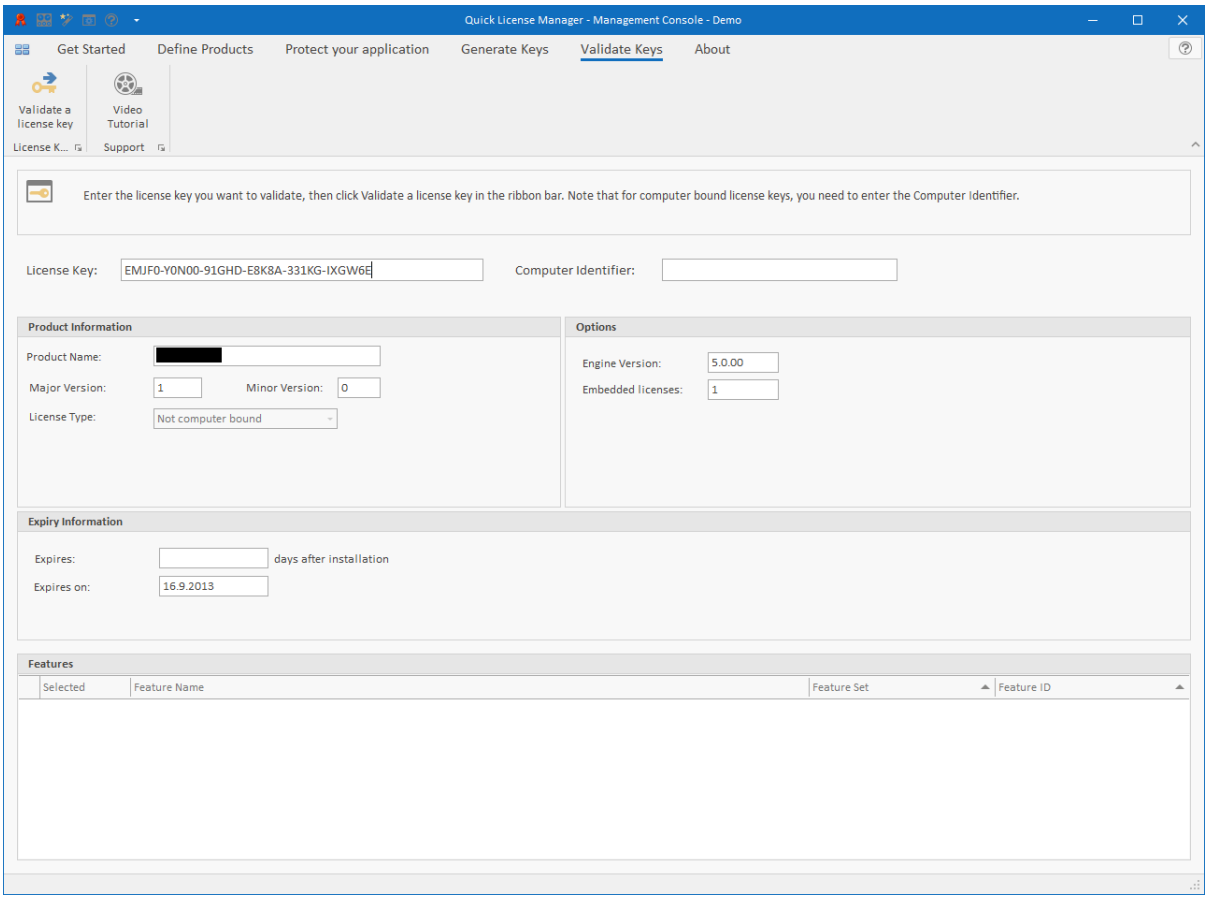
Lisenssille voi myös valita tyypin. Tyyppejä on neljä, jotka ovat Tietokoneeseen sitoutumaton (Not Computer Bound), Tietokoneeseen sidottu (Bound to Computer Name), Käyttäjän määrittelemä (User Defined) ja Aktivointiavain (Activation Key). Tietokoneeseen sidottu ja Käyttäjän määrittelemä avain vaativat nimen, salasanan tai muun syötteen tarkistustekstiksi, jonka avulla sovellus tarkistaa avaimen kelpoisuuden. Esimerkiksi tietokoneeseen sidottuun avaimeen voi laittaa tarkistustekstiksi tietokoneen nimen, jolloin lisenssiavainta tarvitseva sovellus tarkistaa avaimen kelpoisuuden käyttämällä sen tietokoneen nimeä, jolle sovellus on asennettu. Tämä toimintaperiaate on

samanlainen käyttäjän määrittelemässä avaimessa. Tietokoneeseen sitoutumaton avain ja aktivointiavain eivät tarvitse tarkistustekstejä, vaan ne toimivat itsenäisesti. Tietokoneeseen sitoutumatonta avainta voidaan käyttää missä tahansa laitteessa, jolle lisensoitu sovellus on asennettu, eli ei ole rajoituksia laitteelle, johon lisensoitu sovellus asennetaan. Aktivointiavain ei avaa sovellusta käyttöön, vaan se sallii käyttäjän kysyä lisenssiavainta. Aktivointiavaimella tarkistetaan esimerkiksi, että mikä sovellus, mikä versio ja mitä toimintoja sillä saadaan. Tämän mukaan annetaan tietokoneeseen sidottu tai käyttäjän määrittelemä lisenssiavain (Soraco Technologies, 2019.)

Laskentasovelluksissa käytetään lisenssiavaintyyppinä Tietokoneeseen sidottu-tyyppiä. Sovelluksen voi asentaa vain yhdelle tietokoneelle ja luotu lisenssiavain toimii vain sillä koneella. Avaimelle asetetaan ajaksi kuukausi laittamalla viimeinen eräpäivä avaimelle. Näillä asetuksilla generoidaan avaimia asiakkaille.

4.2.6 Avainten vahvistus

Quick License Managerissa on vielä kuvassa 50 näkyvä Vahvista avaimia-sivu, jossa pystyy tarkistamaan luotuja lisenssiavaimia ja niiden sisältämiä tietoja. Sivulla tulee syöttää tarkistettava lisenssiavain ja tarkistusteksti, jos avain sitä tarvitsee, jonka jälkeen sivu näyttää avaimeen kuuluvat tiedot. Nämä tiedot jaetaan neljään kategoriaan, jotka ovat tuotetiedot, asetukset, voimassaolotiedot ja toiminnot.



KUVA 50. Avainten vahvistus.

Tuotetiedoista löytyvät tuotteen nimi, versio ja lisenssityyppi. Asetuksista löytyvät moottoriversio, jolla avain on luotu, ja avaimeen sisältyvien lisenssien määrä. Voimassaolotiedoista löytyvät avaimen voimassaoloaika ja tieto siitä, meneekö avain umpeen tiettynä päivämääränä vai tietyn ajan päästä. Toiminnot ovat toimintoja, jotka kyseinen avain avaa lisensoitavasta sovelluksesta. Näin voi luoda avaimia, jotka avaavat vain tiettyjä osia sovelluksesta. Toimintoja voi luoda vain Quick License Manager Pro- ja Enterprise-versioissa, joten sitä ei käydä tarkemmin läpi.

4.2.7 Muutokset ja lisäykset ohjelmistoihin

Koska QLM:n luoma LicenseValidator.vb ei sopinut laskentasovellusten lisensoinnin tekoon, piti ottaa QLM:n mukana tulleesta Express-version VB.NET esimerkkiprojektista sen versio LicenseValidator.vb-tiedostosta. Tiedoston tarkoitus on toimia rajapintana sovelluksen ja QlmLicenseLib.dll-koodikirjaston välillä. QlmLicenseLib.dll sisältää kaikki lisensointiin kuuluvat toiminnot ja tarkistukset, ja se liitetään jokaiseen laskentaohjelman lähdekoodiin viitteeksi.

Lähdekoodeihin tehdyt muokkaukset voidaan jakaa kolmeen osaan: Lisenssin tarkistus ohjelman käynnistyessä, lisensointi-ikkuna ja LicenseValidator.

4.2.7.1 Lisenssin tarkistus

Kun laskentasovellus käynnistetään, ensimmäinen asia mitä tulee tehdä, on tarkistaa, onko sovellukselle olemassa lisenssiavainta. Tämä tehdään tarkistamalla lisenssin olemassaoloa tietokoneelta, jolle laskentasovellus on asennettu.

```
'Tarkistetaan lisenssi
'Laitetaan muuttujaan uusi LicenseValidator
license = New QLM.LicenseValidator()

If VerifyLicense() = False Then 'Tarkistetaan onko koneella tarvittava lisenssi
    'funktiolla VerifyLicense(), joka palauttaa joko True tai False. Jos tulee False, ohjelma suljetaan

    Environment.Exit(0)

End If
```

KUVA 51. Lisenssin tarkistaminen.

Kuvan 51 koodi löytyy ensimmäisenä avautuvan käyttöliittymän Load-funktiosta, eli se on ensimmäinen asia, jonka ohjelma ajaa käynnistyessään. Käyttöliittymän koodin alussa ennen funktioita luodaan QLM.LicenseValidator-tyyppinen muuttuja nimeltä license. Tämä alustetaan Load-funktiossa. Seuraavaksi koodi kutsuu VerifyLicense-funktiota ja odottaa, mitä se palauttaa. Jos se palauttaa True, lisensointi on kunnossa ja koodi jatkaa eteenpäin normaalilla toiminnalla. Jos se on False, lisensointi ei ole kunnossa ja sovellus sulkee itsensä.

Lisenssin tarkistus aloitetaan siirtymällä funktioon nimeltä VerifyLicense. VerifyLicensen voi jakaa kahteen osaan. Ensimmäisessä osassa luodaan uniikista tunnisteesta avain, jota käytetään tietokoneen tunnisteena. Tämä avain luodaan eräästä tietokoneen tietoihin liittyvästä muuttujasta. Alun perin suunniteltiin käytettävän tietokoneen nimeä kyseisen muuttujan tilalla, mutta koska se on helposti vaihdettavissa, päädyttiin siihen lopputulokseen, että se olisi luonut riskin lisenssin toimivuudelle. Samasta syystä myös tarkkaa käytettävää muuttujaa ei paljasteta tässä raportissa.

Koodissa muuttuja etsitään käyttämällä ManagementObjectSearcher-tyyppistä muuttujaa, jonka avulla etsitään tietokoneen systeemistä kyseessä oleva muuttuja. Tämä muuttuja tallennetaan String-tyyppiseen muuttujaan. Jos tietokoneelta ei löydy kyseistä muuttujaa, koodi palauttaa tyhjän arvon, tai muu virhe tapahtuu haussa, tulee vakiona muuttujaan arvoksi tietokoneen nimi. Kun muuttujaan on saatu arvo, se tiivistetään kuvan 52 koodia käyttäen.

```
'Tiivistetään saatu muuttuja computerID-muuttujaan. Tämä piilottaa käyttäjälle mitä tietoa käytetään tietokoneen tunnistamiseen

'Käyttäjälle näytetään LicenseValidationFormissa tiivistetty muuttuja,
'jonka hän lähettää ohjelman tarjoajalle, joka luo lisenssiavaimen tämän tiivistetyn muuttujan avulla

Dim tmpSource() As Byte
Dim tmpHash() As Byte
tmpSource = ASCIIEncoding.ASCII.GetBytes(Muuttuja)
tmpHash = New MD5CryptoServiceProvider().ComputeHash(tmpSource)
Dim i As Integer
Dim sOutput As New StringBuilder(tmpHash.Length)
For i = 0 To tmpHash.Length - 1
    sOutput.Append(tmpHash(i).ToString("X2"))
Next
computerID = sOutput.ToString()
```

KUVA 52. Muuttujan tiivistäminen.

Tiivistämistä käytetään useimmiten salasanojen sekoittamiseen, ja siksi siihen löytyykin valmiita kirjastoja ja funktioita. Tiivistämisen tarkoituksena on ottaa saatu muuttuja ja muuttaa se muotoon, josta käyttäjä ei tunnista sen alkuperää. Tämä tiivistettynä ulostuleva teksti tallennetaan muuttujaan computerID, ja sitä käytetään lisensoinnissa tietokoneelle räätälöidyn lisenssiavaimen luomiseen ja tarkistamiseen. (Greenberg, 2016.)

Kun computerID on määritelty, aloitetaan tarkistusprosessi. Tämä suoritetaan Do-While-silmukan sisällä, eli silmukka toistuu, kunnes sen ehtolause täyttyy tai jos siitä poistutaan manuaalisesti. Ehtolauseessa tarkistetaan kuvan 53 funktiota license.ValidateLicenseAtStartup, joka sijaitsee LicenseValidator.vb-tiedostossa. Sinne lähetetään computerID parametrina kahden kovakoodatun arvon, needsActivation ja errorMessagen, kanssa. NeedsActivationin arvo on false ja errorMessagelle annetaan tyhjä arvo. Funktio palauttaa arvon True, jos tietokoneella on aktivoitu lisenssi ja muussa tapauksessa palautetaan False.

```
'Tarkistetaan lisenssiavain loopin sisällä, kunnes saadaan selville avaimen tilanne
Do While license.ValidateLicenseAtStartup(computerID, needsActivation, errorMessage) = False

    'Jos avainmen aika on ummessa poistetaan avain ja ilmoitetaan siitä
    If license.EvaluationExpired = True Then
        license.DeleteKeys() 'Poistetaan päättynyt avain koneelta
        MsgBox("Sinun lisenssisi on loppunut! Ota yhteys sovelluksen tarjoajaan saadaksesi uuden lisenssiavaimen.", MsgBoxStyle.OkOnly, "Huomio!")
    End If

    'Jos avainta on peukaloitu tai systeemin kelloa laitettu taaksepäin, tämä laukeaa LicenseValidatorissa, ja muuttuu TamperingDetected = true
    If TamperingDetected = True Then
        MsgBox("Tunnistimme peukaloidun lisenssiavaimen. Jos olet kääntänyt tietokoneen kelloa taaksepäin,
            voit jatkaa sovelluksen käyttöä kun olet asettanut oikean päivämäärän takaisin.", MsgBoxStyle.OkOnly, "Huomio!")
        End
    End If

    'Jos lisenssirekisteröinti-ikkuna suljetaan eikä kunnollista avainta ole lisätty, laitetaan VerifyLicense = false, ja poistutaan funktiosta
    If DisplayLicenseForm(computerID) = DialogResult.Cancel Then
        VerifyLicense = False
        Exit Function
    End If
Loop
```

KUVA 53. Avaimen tilanteen tarkistus.

Kuvan 53 silmukka alkaa aina tarkistamalla lisenssin tilanne ValidateLicenseAtStartup-funktiolla. Silmukan sisälle mennään, jos koodi ei löydä aktivoitua lisenssiä tietokoneelta. Silmukan tarkoituksena on kysyä käyttäjältä lisenssiavainta ja tarkistaa sitä niin kauan, kunnes ehtolauseen ehto ei toteudu (eli tietokoneelta löytyy lisenssiavain), tai jokin silmukan sisällä olevista ehtolauseista toteutuu. Ensimmäisessä ehtolauseessa tarkistetaan, onko lisenssiavaimen aika mennyt umpeen tarkistamalla LicenseValidator-tiedoston kautta EvaluationExpired-muuttujan arvo. Jos tämä arvo on True, eli avain on mennyt umpeen, poistetaan lisenssiavain tietokoneelta ja ilmoitetaan siitä käyttäjälle.

Seuraavan ehtolauseen tarkoitus on tarkistaa, onko avainta peukaloitu tai tietokonesysteemin kelloa säädetty siinä toivossa, että avaimen päivämäärärajoituksia voisi ohittaa. QlmLicenseLib.dll-tiedosto sisältää tarvittavat funktiot peukaloinnin tarkistamiseen, joten LicenseValidator.vb-tiedostossa tarvitsee vain tarkistaa, että onko sen arvo True. Jos se on True, asetetaan sen arvo ensimmäisenä avattavassa käyttöliittymässä sijaitsevan muuttujan TamperingDetected arvoksi. Jos TamperingDetected on True, näytetään käyttäjälle viestilaatikko, jossa kerrotaan peukaloinnista ja annetaan ohjetta, miten sen voi korjata kääntämällä tietokoneen kellon takaisin oikeaan aikaan. Korjaus pätee tilanteisiin, jossa käyttäjä on vahingossa kääntänyt tietokoneen kelloa eteen- tai taaksepäin. Kun viesti suljetaan, laskentasovellus sulkee itsensä automaattisesti.

Viimeinen ehtolause on tärkein ehtolauseista, sillä siinä avataan käyttöliittymä lisenssin syöttämistä varten. Tämä tehdään kuvan 54 DisplayLicenseForm-funktiolla.

```
Private Function DisplayLicenseForm(ByVal computerID As String) As DialogResult

    'Avataan lisenssirekisteröintiformi ja lähetetään sinne muuttujat license ja computerID
    Dim licenseFrm As LicenseValidationForm
    licenseFrm = New LicenseValidationForm(license, computerID)

    'Otetaan palautettavaksi formista tuleva DialogResult
    DisplayLicenseForm = licenseFrm.ShowDialog()

End Function
```

KUVA 54. DisplayLicenseForm-funktio, jolla avataan käyttöliittymä lisenssin syöttöön.

Funktion avulla avataan lisenssin syöttöön tarkoitettu käyttöliittymä ja otetaan arvoksi takaisin DialogResult-tyyppinen vastaus. Tämän vastauksen avulla näkee, onnistuiko lisenssiavaimen syöttäminen. Jos vastauksena tulee DialogResult.Cancel, niin lisenssin luonnin katsotaan epäonnistuneen, jolloin poistutaan silmukasta.

Kun silmukasta pääsee ohi, on saatu arvo muuttujalle VerifyLicense. Jos silmukasta pääsee ohi joko suoraan tai onnistuneen lisenssiavaimen syötön jälkeen, tulee VerifyLicense-muuttujan arvoksi True. Muuten arvoksi tulee False. False-arvo asetetaan, jos DisplayLicenseForm-funktion vastauksena tulee DialogResult.Cancel. VerifyLicensen arvo palautetaan takaisin aloituskäyttöliittymän Load-funktioon, jonka jälkeen arvon avulla jatketaan eteenpäin sovelluksen toiminnassa.

4.2.7.2 Rekisteröinti

DisplayLicenseForm-funktiossa kutsutaan uutta käyttöliittymää nimeltä LicenseValidationForm. Käyttöliittymän tehtävänä on antaa käyttäjälle keino syöttää lisenssiavain sovellukseen ja nähdä lopputulokset sille. Sivulla löytyy myös ohje avaimen hankkimiseen.

KUVA 55. Lisensointi-ikkuna.

Kuvan 55 käyttöliittymässä on neljä tekstikenttää. Ylimpään kirjoitetaan lisenssiavain, kahdella alemmalla näytetään tietoa lisenssiavaimen rekisteröinnin lopputuloksesta ja vasemmalla näkyy tiivistetty tunnistemuuttuja, eli computerID-muuttujan arvo. Tämä teksti pitää lähettää sovellusten omistajalle ja sen avulla luodaan käyttäjälle lisenssiavain.

Kun käyttöliittymä avataan, ottaa käyttöliittymä vastaan tietoja aloituskäyttöliittymältä. Nämä tiedot ovat computerID-muuttuja ja license-muuttuja. Kun käyttäjä syöttää avaimen ja painaa Rekiste-

röidy-nappia, käyttöliittymän koodissa otetaan syötetyt tiedot talteen ja kutsutaan funktiota Register. Funktion sisällä kutsutaan license-muuttujan avulla funktiota ValidateLicense tiedostosta LicenseValidator, minkä avulla rekisteröidään avain tietokoneelle, jos se on oikein. Funktio palauttaa ja asettaa lisenssikäyttöliittymän tekstikenttiin viestejä avaimen rekisteröinnin lopputuloksen mukaan ja asettaa arvon muuttujalle licenseIsValid. Muuttujan tarkoitus on tiedottaa, että onnistuiko rekisteröinti. Kun käyttäjä painaa Sulje-nappia, lisensointikäyttöliittymä tarkistaa licenseIsValid-muuttujan avulla, mitä palautetaan tietona takaisin aloituskäyttöliittymälle. Jos licenseIsValid on False, palautetaan DialogResult.Cancel. Jos taas arvoksi saadaan True, palautetaan DialogResult.OK.

4.2.7.3 LicenseValidator-luokka

LicenseValidator-tiedoston tarkoitus on toimia koodimuotoisena käyttöliittymänä sovelluksen ja QlmLicenseLib.dll-tiedoston välillä. Tiedoston tehtävänä on tarkistaa lisenssiavaimien kelpoisuus ja toimia sen mukaisesti. Normaalisti Quick License Manager loisi nämä tiedostot automaattisesti, mutta luodut tiedostot sisälsivät funktioita, jotka haittasivat lisenssintarkistusta ja estivät sitä. Ratkaisuna tähän oli ottaa QLM:in mukana tulleista esimerkkiprojekteista niiden LicenseValidator-tiedosto ja muokata se siihen muotoon, jota laskentasovellukset tarvitsivat.

LicenseValidator-tiedostoilla ei ole eroja eri laskentasovellusten välillä, muuten kuin tuotetietojen määrittelyn kanssa. Seuraava koodi luetaan aina, kun alustetaan uusi QLM.LicenseValidator-muuttuja lisensointia varten.

```
_license = New QlmLicenseLib.QlmLicense

'HUOMIO: Tähän kirjoitetaan tuotteen tiedot miten ne on luotu Quick License Managerissa
'DefineProduct(): Tuotteen ID, Tuotteen nimi, pääversio, minor versio, enkryptioavain (jätetään aina tyhjäksi), tuotteen GUID
'PublicKey: Tuotteen public key QLM:stä

_license.DefineProduct(5, "JbPalkki EC2 Pro", 1, 0, "", "{<GUID>}")
_license.PublicKey = "<julkinen avain>
```

KUVA 56. Tuotteen tietojen asettaminen LicenseValidator-tiedostossa.

Kuvan 56 LicenseValidator-tiedostossa on sen tuotteen tiedot, jotka vastaavat sitä laskentasovellusta, jossa tiedosto on osana. Nämä tiedot ovat:

- Tuotteen ID
- Tuotteen nimi
- Tuotteen isompi versionumero
- Tuotteen pienempi versionumero
- GUID
- Julkinen avain

Nämä tiedot julkista avainta lukuun ottamatta on manuaalisesti kovakoodattu funktioon DefineProduct. Julkinen avain on määritelty muuttujassa PublicKey. Esimerkistä on poistettu GUID ja julkinen avain turvallisuussyistä.

DefineProduct-funktioon syötetään tiedot samassa järjestyksessä, jossa ne on esitelty aikaisemmassa listassa. Funktiossa on tuotteen pienemmän versionumeron ja GUID:n välissä tyhjä arvo, jota funktio tarvitsee, mutta siihen ei ole tarpeellista syöttää mitään, joten se jätetään tyhjäksi.

LicenseValidator-tiedosto lisättiin jokaisen laskentasovelluksen lähdekoodiin siten, että niihin kopioitiin pohjatiedosto, jossa ei ole vielä tuotetietoja. Sen jälkeen lisättiin tiedostoihin laskentasovelluksia vastaavat tuotetiedot. LicenseValidatorin pohjatiedostoon lisättiin myös kutsu laskentasovellusten aloituskäyttöliittymään muuttujalle TamperingDetected. Aina kun tiedostossa tarkistetaan, että onko lisensointia peukaloitu, ja se palauttaa True, lähetetään tämä arvo aloituskäyttöliittymässä olevalle muuttujalle, jonka avulla lopetetaan sovelluksen käyttö välittömästi.

4.2.8 QLM:n käyttö

Quick License Managerin ja laskentasovellusten lisensointiosuuden käyttö on suhteellisen yksinkertaista. Työläimmät tehtävät on suoritettu lisensointitoimintojen rakentamisen ja lisäämisen myötä, joten arkipäiväiseen käyttöön liittyy vain avainten luonti, lähettäminen ja arkistointi.

Laskentasovelluksissa käytetään lisenssiavaintyyppinä Tietokoneeseen sidottu-tyyppiä. Sovelluksen voi asentaa vain yhdelle tietokoneelle ja luotu lisenssiavain toimii vain sillä koneella (Soraco Technologies, 2019). Avaimelle asetetaan ajaksi kuukausi laittamalla viimeinen eräpäivä avaimelle. Ongelmana tuli vastaan ainoastaan se, että mikä olisi tietokoneille uniikki tunniste. Tietokoneen nimeä ei voi käyttää, sillä kuka tahansa voi sitä vaihtaa helposti. Tietokoneen osien tunnisteita ei voida käyttää, sillä osat saattavat vaihtua, jolloin sovellus lakkaisi toimimasta, joten lopulta päädyttiin käyttämään erästä tietokoneeseen sidottua muuttujaa. Tunniste tiivistetään, jottei käyttäjä tiedä, mistä tunniste on peräisin. Tämä tiivistetty tunniste näytetään lisenssiavainta kysyessä.

Toimintaperiaate lisenssin aktivoinnille on, että kun asiakas ensimmäisen kerran käynnistää laskentasovelluksen, avautuu lisenssiavaimen rekisteröinti-ikkuna. Ikkunasta löytyy ohje avaimen hankkimiselle ja tiivistetty muuttuja. Jotta avaimen saa, täytyy käyttäjän lähettää sähköpostia sovelluksen omistajalle ja liittää siihen mukaan sovelluksen näyttämä tiivistetty muuttuja. Kun omistaja saa sähköpostin, pystytään luomaan asiakkaan tietokoneeseen sidottu lisenssiavain, jonka tarkistusteksti on tiivistetty muuttuja. Kun avain on luotu, tarkistettu ja arkistoitu, lähetetään se asiakkaalle sähköpostitse. Asiakas käyttää avainta rekisteröidäkseen lisenssin, minkä jälkeen sovellusta voi sitten käyttää vapaasti sen ajan, joka on avaimeen asetettu.

4.3 Lisensoinnin lopputulos

Kaikki lisensointiin liittyvät tavoitteet saavutettiin. Sovellukset tarvitsevat lisenssiavaimen toimiakseen ja sovellukset toimivat vain sillä tietokoneella, jolle avain on luotu ja vain tietyn ajan. Saavute-

tut tavoitteet kuitenkin eroavat alkuperäisistä tavoitteista, joihin kuului kelluva lisenssi ja lisenssipalvelin, mutta nämä todettiin epäkäytännöllisiksi ja liian kalliiksi ylläpitää yhdessä toimeksiantajan kanssa.

Lisensoinnin mukana luotiin ohjeistusdokumentin, jonka avulla pystyy lisensointitoiminnon lisäämään uusiin sovelluksiin, jos sellaisia koskaan tehdään. Mukaan on lisätty myös kaikki tarvittavat tiedostot ja esimerkkikuvia. Tiedostoihin kuuluvat kaikki lisenssin rekisteröinti-ikkunan tiedostot (LicenseValidationForm.Designer.vb, LicenseValidationForm.resx, LicenseValidationForm.vb), LicenseValidator.vb ja kopio QlmLicenseLib.dll-tiedostosta. Tiedostoihin kuuluu myös tekstitiedostoja, jotka sisältävät koodiotteet, jotka puolestaan kopioidaan uuden sovelluksen lähdekoodiin.

Lisensointia ei tarvitse tulevaisuudessa päivittää, kunhan Quick License Manager ei saa niin laajaa päivitystä, joka vaatii lisenssiavainmoottorin vaihtamista tai QlmLicenseLib.dll-tiedoston vaihtamista.

5 JATKOKEHITYS JA POHDINTA

5.1 Pohdinta – Eetu Häkkinen

Työssä riitti tekemistä kahdelle ihmiselle mielestäni sopivasti, ja sen sisältö oli tarpeeksi vaihtelevaa pitämään työskentelyn mielenkiintoisena. Kaikista vastaan tulleista ongelmista pääsimme yli, oppien samalla uutta.

Aikataulut pitivät lähes täysin paikkansa ja työn migraatio etenikin odotettua vauhtia. Pientä viivästymistä aiheutui työn loppuvaiheessa, kun oli asiana selvittää paras metodi lisensoinnille, mutta tämäkään ei hidastanut työtä paria viikkoa enempää. Työtä testasimme itse niin paljon, kuin pystyimme, mutta laskentaohjelman monimutkaisuuden takia syvällisemmän testauksen jätimme työn tilaajalle. Näin myöskään laskennan testaus itsessään ei vienyt liian paljon aikaa, vaan pääsimme keskittymään migraatioon.

Työn aikana pidimme yhteyttä viikoittaisilla päivityssähköposteilla, joiden lisäksi lähetimme myös sähköposteja kiireellisimpien asioiden suhteen. Tapaamisia pidettiin tarpeen vaatiessa ja yhteistyö onnistuikin hyvin, sillä tavoitteista päästiin helposti yhteisymmärrykseen.

Eniten itse opin lisää Visual Basicin 6.0- ja .NET- versioista, sekä WinFormsin paremmasta käyttämisestä. Myös työn etenemisen kirjaaminen ja oman koodin kommentointi nousi erittäin tärkeäksi työn aikana.

5.2 Pohdinta – Valtteri Kovalainen

Opinnäytetyö oli loistava kokemus ja tilaisuus oppia työelämän tavoista, projektityöskentelystä ja kommunikaatiosta. Aluksi ei oltu varmoja, että riittäisikö työtehtäviä opinnäytetyöksi asti, mutta kun niitä kartoitettiin ja suunniteltiin enemmän, saatiin tästä hyvin kahdelle henkilölle työtä. Tavoitteita ja työtehtäviä jäi ylikin, joten se antaa erinomaisen tilaisuuden jatkotyöskentelylle opinnäytetyön jälkeen.

Toimeksiantaja Ari Korhosen ja Eetu Häkkisen kanssa työskenteleminen oli erittäin sujuvaa ja mieluisaa. Kommunikaatiota suoritettiin viikoittaisilla sähköposteilla ja epäsäännöllisillä palavereilla, joiden avulla tiedotimme toisiamme tarkasti. Käytimme myös runsaasti sisäistä kommunikaatiota Eetu Häkkisen kanssa, jonka avulla suunnittelimme ja toteutimme opinnäytetyötä.

Onnistuimme pitämään sovitusta aikataulusta kiinni, ja tavoitimme kaikki sovitut päätavoitteet, eli migraation, lisensoinnin ja asennusohjelman luonnin. Jouduimme tekemään muutamia kompromisseja joidenkin ratkaisujen kanssa, mutta saadut lopputulokset täyttävät tavoitteiden vaatimukset.

Opinnäytetyö opetti minulle uuteen työprojektiin tutustumista ja mukautumista, sillä minun piti opetella Visual Basicin käyttöä ja perehtyä tuntemattomaan lähdekoodiin mahdollisimman nopeasti. Tämä kokemus tulee auttamaan minua työurallani, joka hyvin todennäköisesti vaatii taitoja ja kokemusta uuteen projektiin sopeutumisesta ja uusien taitojen opettelemisesta. Keskityimme myös erittäin paljon koodin ja toimintojen kommentointiin ja dokumentointiin, joten saimme loistavaa kokemusta sen suhteen. Kattavaa dokumentaatiota tarvitaan, jotta toimeksiantaja voi jatkokehittää sovelluksia helpommin.

5.3 Jatkokehitys

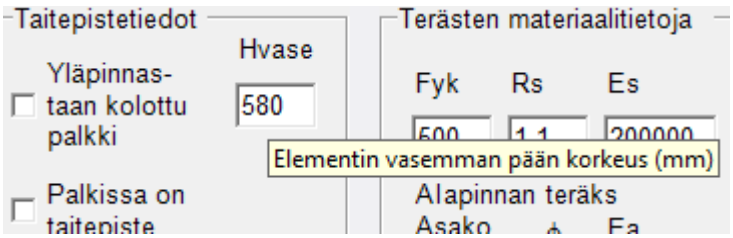
Sovellukseen todettiin tarpeelliseksi lisätä vielä monta toimintoa, mutta se ei ollut mahdollista opinnäytetyön aikarajoitteiden puitteissa. Tästä riittäisi todennäköisesti tekemistä toiseen opinnäytetyöhön tai harjoitteluun, mikä on todennäköisesti järkevä tapa hoitaa kehitys, sillä kehittäjien palkkaaminen ei ole kannattavaa ohjelmistojen tämänhetkisellä kysynnällä.

Tarkoituksena on pitkälti EC2-ohjelmien, eli uudempaa EU-standardia seuraavien, jatkokehitys, sillä suomen vanhempaa standardia käyttäviä ohjelmistoja ei myydä ulkomaille.

5.3.1 Käännösmahdollisuus

Jos ohjelmistoja missään vaiheessa myydään ulkomaille, on niiden suomenkielisyys este. Siksi onkin tärkeää kehittää metodi, jolla sovellukseen pystytään dynaamisesti lisäämään uusia kieliä tarpeiden puitteissa. Tähän alustavasti olisi tarkoituksena käyttää jotakin tiedostoa, jonka käyttäjä pystyy itse avaamaan. Näin käyttäjä pystyy myös kääntämään ohjelmiston englannista omalle kielelleen, jos osaamista ja halua löytyy. Lisäksi ohjelmaan olisi suotavaa koodata metodi tiedostojen palautukselle, jos käyttäjä poistaa tai rikkoo ne.

Tarkoitus on lisätä ohjelmaan englanti toissijaiseksi kielivalinnaksi, johon voi vaihtaa milloin tahansa käytön aikana tai jo sovellusta rekisteröidessä. Käyttäjiä voidaan myös ohjeistaa jakamaan oma käännöksensä takaisin Laaturakenteelle, jos omia käännöksiä tehdään. Tätä voidaan käyttää useampien kielien lisäämiseen ohjelmistoon, eli onkin tärkeää, että kieliä voi lisätä ainakin teoriassa loputtomasti.

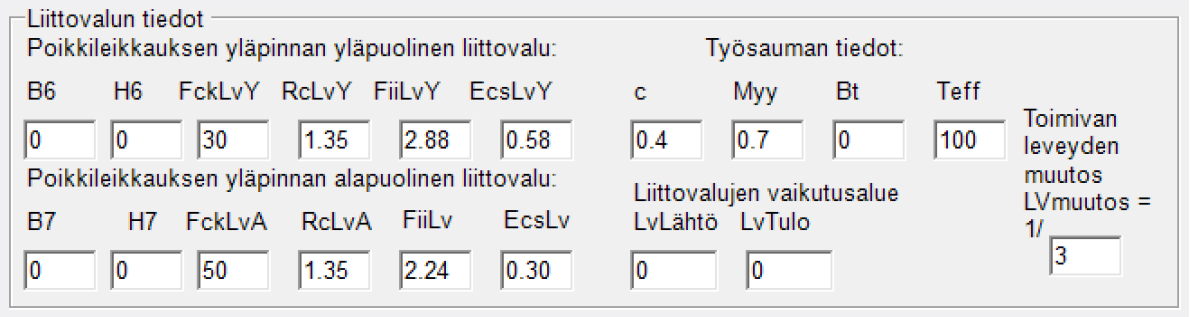


KUVA 57. Esimerkki vinkistä, joka näytetään pidettäessä hiirtä kentän päällä.

Ohjelmistoissa on kaikissa paljon käännettävää, joten tarkoitus olisikin aloittaa uusimmasta ja laajimmasta ohjelmasta, jossa työtä on eniten. Varsinkin vinkeissä, joista esimerkki voidaan nähdä kuvassa 57, on erityisen suuri ja aikaa vievä työ tehtävänä. Tähän työhön perustaen pitäisi olla suhteellisen helppoa lisätä samat muutokset suppeampiin ohjelmistoihin. Vaatimusten puitteissa kieltä pitäisi pystyä vaihtamaan lennossa, eli ohjelmaa ei joutuisi käynnistämään uudelleen kielen vaihtamiseksi.

5.3.2 Muokattavat vakioarvot ja -kertoimet

Kansainvälisille markkinoille pääsyssä on myös tärkeää mahdollistaa tiettyjen kertoimien ja arvojen oletusarvojen muokattavuus. Vaikka EU-maat seuraavatkin samoja laskentakaavoja ja -periaatteita, nämä kertoimet saattavat maiden välillä vaihdella. Jos käyttäjä joutuu joka käyttökerralla muuttamaan kertoimet manuaalisesti, on käyttäminen hankalaa ja hidasta, kuten ilmenee kertoimien paljoudesta kuvassa 58. Tarkoituksena olisi siis lisätä uusi toiminto, jolla käyttäjä voi luoda uudet oletusasetukset, jotka ladataan normaalien asetusten sijaan ohjelman käynnistyessä.



KUVA 58. Joitakin kertoimia, joiden oletusarvoja olisi tarkoitus muuttaa.

Myös tässä toiminnossa on erityisen tärkeää muistaa lisätä mahdollisuus oletusasetusten palauttamiselle, jos käyttäjä haluaa palata takaisin alkuarvoihin, eikä muista tai tiedä niitä. Toisin kuin kielen vaihdossa, uusien asetusten lataaminen tulee vaatimaan ohjelman uudelleenkäynnistyksen. Näin estetään tilanne, jossa käyttäjä vahingossa muuttaa auki olevia laskelmiansa ja menettää mahdollisesti usean tunnin työn.

5.3.3 Laskennan automaattinen testaus

Koska kyseessä olevia ohjelmistoja käytetään rakennusten tukirakenteiden lujuuden laskemiseen, on ensisijaisen tärkeää, että laskelmiin ei ilmene pienintäkään virhettä. Tämän takia ohjelmistoja onkin laajasti testattava jokaisen päivityksen jälkeen, mihin kuluu paljon aikaa. Manuaalinen testaaminen tuo myös mahdollisuuden inhimillisille virheille, joka ei ole suotavaa.

Tarkoituksena olisikin luoda testausympäristö ainakin uusimmalle ohjelmistolle, jotta laskennan testauksen voisi automatisoida. Näin kehityksessä säästettäisiin erittäin paljon aikaa ja vältyttäisiin mahdollisilta virheiltä. Tämä on kuitenkin tärkeysjärjestyksessä viimeisenä jatkokehitettäessä, sillä suurinta osaa ohjelmista ei laskennan osalta kehitetä enää eteenpäin.

6 LÄHDELUETTELO

10Duke. (2019). *What is a floating software licence?* (10Duke) Haettu 23. Heinäkuu 2019 osoitteesta 10duke.com: https://www.10duke.com/blog/floating-software-licence/

Empira Software GmbH. (11. Elokuu 2009). *PDFsharp & MigraDoc*. (Empira Software GmbH) Haettu 23. Heinäkuu 2019 osoitteesta http://www.pdfsharp.net: http://www.pdfsharp.net/MigraDocOverview.ashx

Empira Software GmbH. (14. Syyskuu 2015). *PDFsharp and MigraDoc Wiki*. Haettu 8. Elokuu 2019 osoitteesta http://www.pdfsharp.net/wiki: http://www.pdfsharp.net/wiki/HelloMigraDoc-sample.ashx

Greenberg, A. (8. Kesäkuu 2016). *WIRED*. (A. Greenberg, Toimittaja) Haettu 8. Elokuu 2019 osoitteesta https://www.wired.com: https://www.wired.com/2016/06/hacker-lexicon-password-hashing/

Microsoft Oy. (20. Heinäkuu 2015). *Numeric Data Types (Visual Basic)*. Haettu 23. Syyskuu 2019 osoitteesta https://docs.microsoft.com: https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/data-types/numeric-data-types

Microsoft Oy. (15. Kesäkuu 2017). *Automatic scaling in Windows Forms*. Haettu 8. Elokuu 2019 osoitteesta https://docs.microsoft.com: https://docs.microsoft.com/en-us/dotnet/framework/winforms/automatic-scaling-in-windows-forms

Microsoft Oy. (2019). *Microsoft Power Packs*. (Microsoft) Haettu 23. Heinäkuu 2019 osoitteesta microsoft.com: https://www.microsoft.com/en-us/download/details.aspx?id=25169

Nibu, T. (20. Syyskuu 2011). *bits and bytes*. Haettu 8. Elokuu 2019 osoitteesta https://ntcoder.com/bab: https://ntcoder.com/bab/2011/09/20/convert-a-vb6-project-to-vb-net-visual-studio-2010/

Russell, J.;& Laan, M. (2019). *Inno Setup*. Haettu 8. Elokuu 2019 osoitteesta http://www.jrsoftware.org: http://www.jrsoftware.org/isinfo.php

Soraco Technologies. (2019). *Compare Editions of QLM's Licensing Service*. (Soraco Technologies) Haettu 23. Heinäkuu 2019 osoitteesta soraco.co: https://soraco.co/quick-license-manager/quick-license-manager-features/

Soraco Technologies. (28. Kesäkuu 2019). *License Key Types*. (Soraco Technologies) Haettu 23. Heinäkuu 2019 osoitteesta support.soraco.co: https://support.soraco.co/hc/en-us/articles/215683603-License-Key-Types

Soraco Technologies. (2019). *QLM Pricing*. Haettu 8. Elokuu 2019 osoitteesta https://soraco.co: https://soraco.co/pricing/details/

Tripathi, S. (15. Joulukuu 2017). *C#Corner*. Haettu 23. Heinäkuu 2019 osoitteesta www.c-sharpcorner.com: https://www.c-sharpcorner.com/blogs/method-overloading-in-c-sharpnet

Verticom Oy. (2018). *Miksi migraatio*. Haettu 8. Elokuu 2019 osoitteesta verticom.fi: https://verticom.fi/asiantuntijapalvelut/migraatio/